DOS EXTRA Nr. 16 – 1991

DOS EXTRA

18,– DM 145 öS 18,– sfr 55,– Fmk

Sonderausgabe DOS EXTRA

Nr. 16 1991

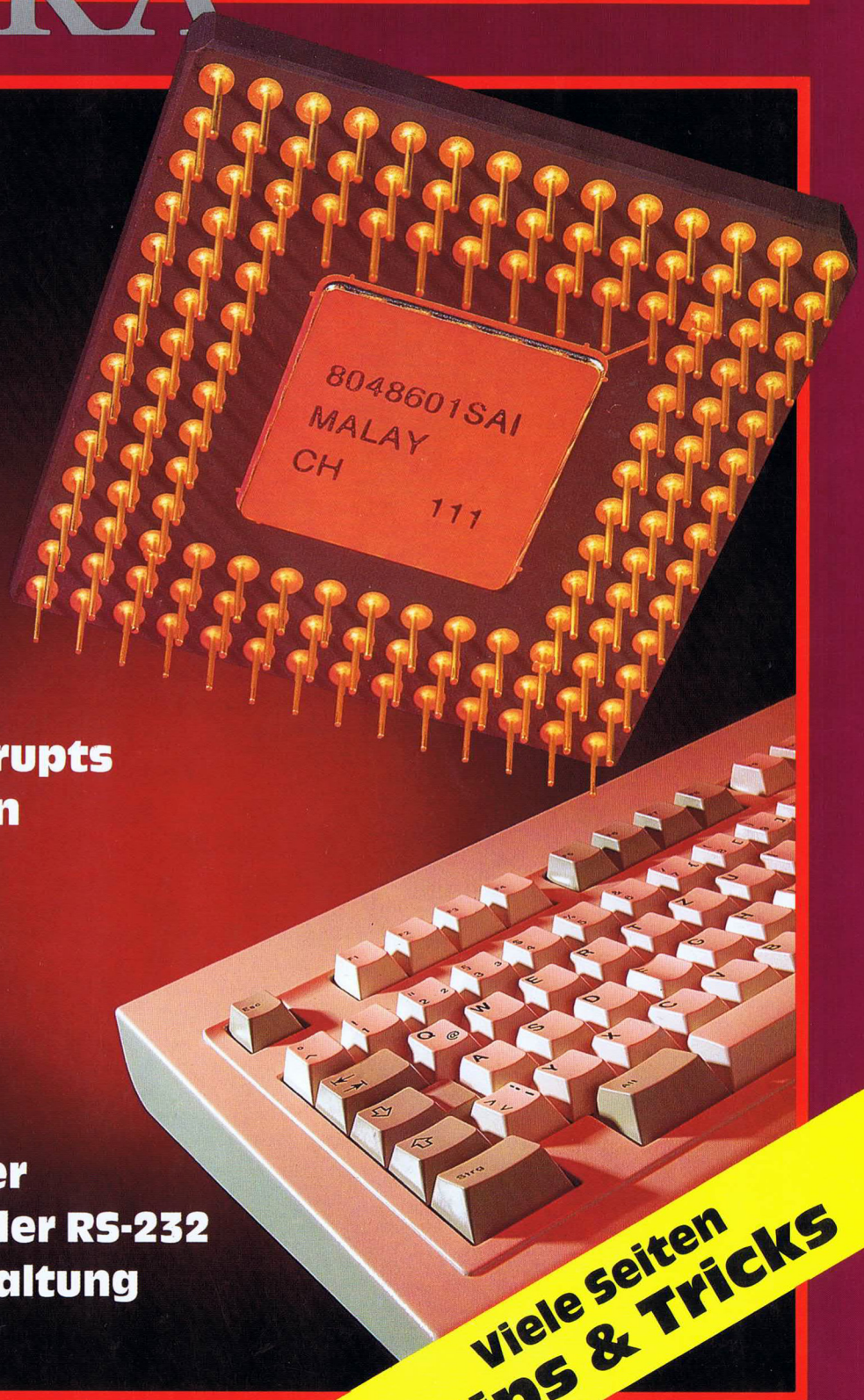
ISSN 0934 – 2842

DMV-Verlag

Eine Publikation der
DOS INTERNATIONAL

Top-Listing:

Compiler für Batchdateien



Assembler

MASM und TASM selbst erweitern

Nachschlagewerk: Befehle und Interrupts

Nützliche Routinen zum Abtippen

PC Intern

Werkstatt: Diskettencontroller

Programmierung der RS-232

XMS-Speicherverwaltung selbstgemacht

Viele Seiten Tips & Tricks

DMV PROGRAMMIERER-TOOLS

Neo Graph

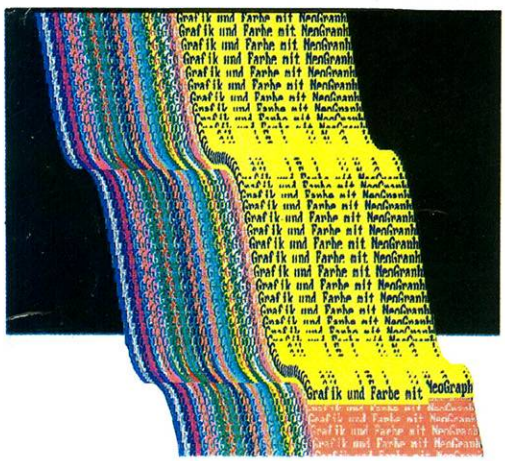
Jetzt ist sie da!!

Die Turbo-Treiber-Toolbox für EGA/VGA-Grafikkarten. Programmierer aufgepaßt!

Turbo Pascal macht bei 640 x 480 Schluß. Hier fängt NeoGraph an! NeoGraph versetzt Sie in die Lage, die Möglichkeiten Ihrer Grafikkarte voll auszunutzen: höchste Auflösungen bei optimaler Farbwiedergabe. Die Programmsammlung NeoGraph stellt ein unverzichtbares Komplettpaket für die Programmierung von Garfikkartentreibern in (Turbo) Pascal dar, insbesondere, wenn es um höhere Auflösung als 640 x 480 geht.

Im Detail:

- NeoGraph enthält einen Satz vorgefertigter Grafikkartentreiber (bis 1024 x 768 bei 256 Farben), zum Beispiel für Tseng, Genoa, Video7, u.a.
- Einige Prototypen von Treibern dienen dazu, auch kommende Formate auszunutzen,
- und um eventuell eigene Sonderformate und Programme anpassen zu können.



- Alle Routinen liegen im Quellcode bei.
- Die Programme sind ausgelegt, um auf DOS-Rechnern in Turbo Pascal ab der Version 4.0 eingebunden zu werden.
- Damit die erweiterten Grafikmodi genutzt werden können, gibt es zwei neue Units: NGRAPH und NCRT!
- Diese Units ersetzen die beiden Original-Units GRAPH und CRT.
- Beide sind voll kompatibel zu den Originalen.
- Beide sind in Pascal geschrieben.
- Beide haben wesentliche Erweiterungen und Verbesserungen gegenüber den Originalen.
- Damit sind sie wesentlich leistungsfähiger.
- Ausführliche Dokumentierung versetzt auch "Normalmenschen" in die Lage, die Programme professionell zu nutzen.
- Drei Testprogramme dienen der Erforschung der Video-Hardware.
- Ein sehr detailliertes Handbuch führt Sie behutsam an die Programmierung heran. Der Autor Arne Schäpers ist bekannt für seine hintergründige aber plausible Art zu schreiben.
- Das kleine Demoprogramm gewährt einen Einblick in die Mächtigkeit dieser einzigartigen Werkzeuge.

NeoGraph **DM 99,—***

GraphBas

BGI-Erweitung für Microsoft Quick Basic & Turbo Basic

GraphBas PC erschließt Ihrem Basic-Compiler bisher ungeahnte Grafikwelten. Was Turbo Pascal und Turbo C berühmt gemacht hat, gibt es jetzt auch für Basic:

- volle Nutzung der BGI-Vektorfonts
- Editor für eigene Schriften, Logos und Pictogramme
- 3D-Balken- und Tortendiagramme, kinderleicht aufgerufen
- Farben, Füllmuster und Figuren mit Assemblerroutinen
- einfachste Handhabung durch Quellcodes-Einbindung
- Unterstützung aller Grafikkarten auf dem PC

GraphBas PC **je DM 99,—***

BGI-Grafikerweiterung für Microsoft Quick oder Turbo Basic

MaskEdit PLUS 5.0

Professioneller Maskengenerator liefert Quellcodes Ihrer Wahl

- statt Listings tippen Menüpunkte auswählen.
- schnelle Bildschirmmasken für Ihre Programme:
- hierachische Menüs
- verschiedene Ausgabefenster
- 8 Rahmentypen, 16 Farben, Schattenwurf
- ASCII-, dBase-, Binär- und ANSI-Ausgabeformate als Standard
- optimaler Treiber für die Ausgabesprache Ihrer Wahl
- alle Treiber im Quelltext
- einfaches, aber effizientes Arbeiten mit über 50 Funktionen
- Komfort durch kontextbezogene Hilfen

Zusatztreiber-Auswahl:

Turbo Pascal 3/4/5, Turbo C und Mircosoft C, Topspeed-Modula-2, Turbo Basic, Quick Basic

MaskEdit Plus, ein Zusatztreiber **DM 99,—***
MaskEdit Plus, zwei Zusatztreiber **DM 149,—***
MaskEdit Plus, drei Zusatztreiber **DM 199,—***

Prof Ed

Der Editor für Programmierer

Ob Sie programmieren, einfach Texte schreiben oder Ihre Config.sys- oder Autoexec.bat-Dateien umändern wollen: ProfEd ist das ideale Werkzeug für Sie:

- Bis zu fünf Dateien lassen sich gleichzeitig bearbeiten.
- Mehrere MByte lange Dateien sind kein Problem mehr.
- Jede Taste der Tastatur kann mit einem oder mehreren Kommandos versehen werden.
- Makros gestatten beliebig viele Editorkommandos abzuarbeiten.
- Mehrere Undo-Funktionen lassen den letzen Arbeitsschritt rückgängig machen.
- Teile einer oder verschiedener Dateien lassen sich gleichzeitig am Bildschirm zur Bearbeitung bereitstellen.
- Suchen und Ersetzen von Zeichenfolgen ist problemlos.
- Zeilen- und spaltenweises Löschen, Kopieren, Verschieben und Füllen von markierten Textbereichen gestatten ein schnelles Bearbeiten Ihrer Dateien. Diese Funktion arbeitet dateiübergreifend und ermöglicht jede nur denkbare Textmanipulation.
- Eine spezielle Netzwerkversion ist erhältlich.

Systemvoraussetzung:
PC, XT, AT, 386er oder 486er Rechner mit mindestens 256 KByte RAM, DOS ab Version 3.0

ProfEd **DM 99,—***

* Unabhängig von der Anzahl der bestellten Programme berechnen wir für das Inland DM 4,— bzw. für das Ausland DM 6,— Porto und Verpackung.
Bitte benutzen Sie die Bestellkarte.

DMV-Verlag · Postfach 250 · W-3440 Eschwege

DMV Daten- und Medienverlag

Eine Frage der Perspektive

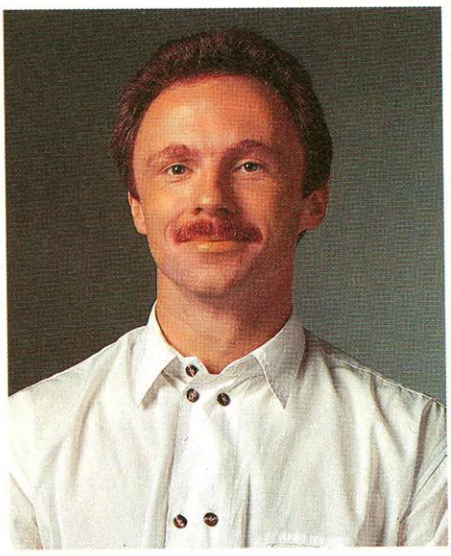
Programmierer genießen in der Welt der Computer einen recht zwielichtigen Ruf. Viele sehen in ihnen etwas weltfremde Wesen, deren Lebensaufgabe darin zu bestehen scheint, dem Rechner immer neue Fähigkeiten zu entlocken. Zu Tages- oder besser gesagt Nachtzeiten, zu denen andere im Land der Träume versinken, dringen sie immer tiefer in die Welt der Bits, Nibbles und Bytes ein. So entstehen anspruchsvolle Programme, die der Anwender bereitwillig nutzt.

Die Anwender sind jedoch nur schwer zufriedenzustellen. Sie verlangen nach immer ausgefeilterer und umfangreicherer Software. »Also, wenn das Programm noch diese kleine Funktion mitbringen würde ...«, ein Spruch, den wohl jeder Programmierer kennt. Doch gerade diese Kleinigkeiten sind meist mit einem erheblichen Entwicklungsaufwand verbunden. Daß MS-DOS den modernen Prozessoren nicht mehr gerecht wird, streitet kaum noch jemand ab. Von den Tücken der Hardware ganz zu schweigen. Und so manche »Kleinigkeit« hat schon einen ganzen Schweif weiterer »Kleinigkeiten« nach sich gezogen.

Nun stellt sich die Frage, ob nicht die Anwender in vielen Fällen zu den Faktoren gehören, die die Programmierer zu endlosen Sitzungen vor dem Rechner treiben. In welcher Gruppe, Anwender oder Programmierer, sind also die »seltsamen Wesen« zu suchen? In der Gruppe derer, die durch ihre Unzufriedenheit immer mehr verlangen, oder bei denen, die die Ansprüche der Unzufriedenen zu erfüllen suchen?

Die Beiträge dieses Heftes richten sich an beide Gruppen. Den Programmierern soll es helfen, die »Kleinigkeiten« ein bißchen schneller in den Griff zu bekommen. Zudem soll es sie etwas entlasten, indem es den Anwendern nützliche und ausgefeilte Helferlein zur Hand gibt. Womit es sich auch an diese richtet. Ob Sie nun Ihre Stapeldateien kompilieren wollen, Ihre »config.sys« während des Startens an die benötigte Treiberkonfiguration anpassen oder Ihre Daten im Erweiterungsspeicher ablegen möchten, hier finden Sie das Richtige. Allen Lesern, zu welcher Gruppe sie auch gehören, wünsche ich viel Spaß mit diesem Heft.

Ralf Morgenstern



INHALT

DOS EXTRA
16/91

ASSEMBLER

Assembler-Präprozessor
Neue Befehle für den Assembler 6

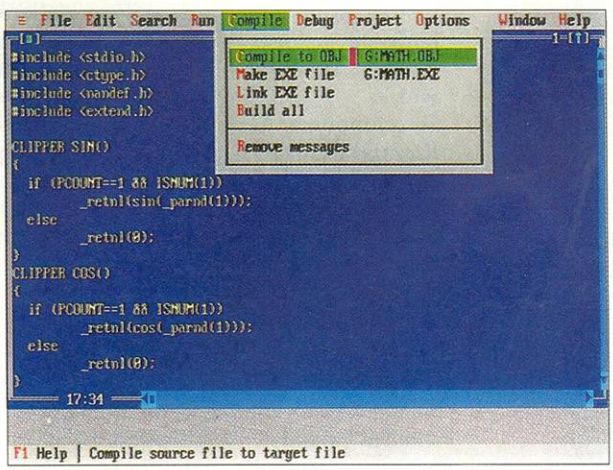
Werkstatt Lib/Tlib
Bilbliotheken für den Linker 16

Grundlagen Grafikkarten
Ein Bild für die Götter 20

System-Bibliothek
Assembler wird gesellschaftsfähig 30

Befehlsreferenz
Die Befehle der Intel-Prozessoren 46

Der Präprozessor erweitert auf einfache Weise den Sprachumfang des Assemblers. Die 40 neuen Befehle finden Sie ab der Seite 6



Wie Sie Objekte mit Bibliotheken-Programmen professionell verwalten erfahren Sie auf Seite 16

Systemprogrammierung

Treiberprogrammierung
Flatterhafte Config-Datei 69

FDC-Werkstatt
Gut in der Spur 76

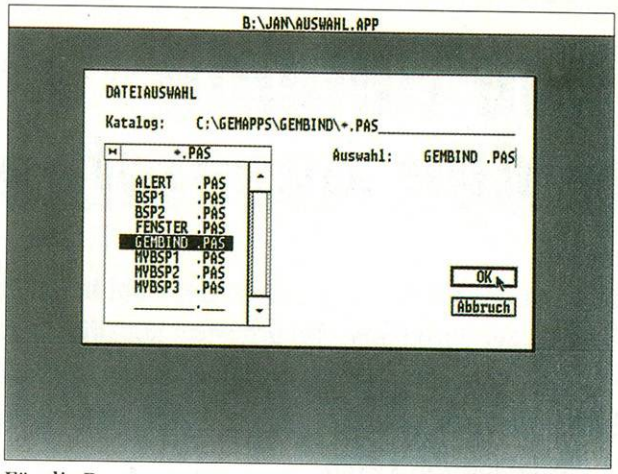
Speicherverwaltung
Daten im Erweiterungsspeicher 86

Programmieren Sie den Controller des Diskettenlaufwerkes direkt. Die FDC-Werkstatt zeigt Ihnen wie Seite 76

GEM-Programmierung
Edelstein für Turbo Pascal 92

Turbo Pascal 6.0
Laufwerke unter der Lupe 106

Interruptcontroller und serielle Schnittstelle
Controller fest im Griff 110



Für die Programmierung von GEM mit Turo Pascal benötigen Sie nur die richtige Unit. Wie es gemacht wird erfahren Sie ab **Seite 92**

TIPS & TRICKS

Tips & Tricks für Programmierer 124

LISTINGS

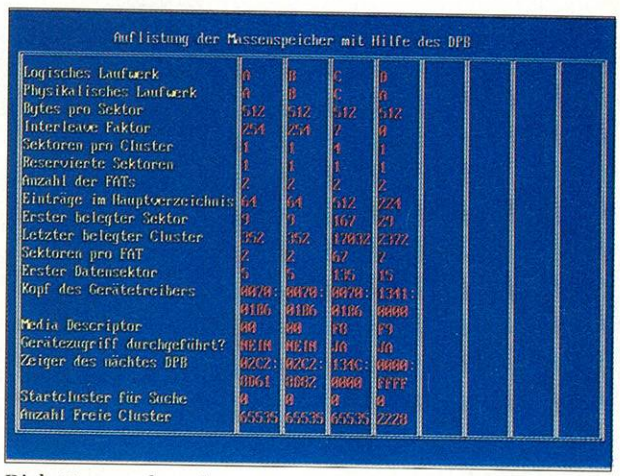
SCheck 3.0
Die nächste Generation 145

Compilerbau
Compiler für Stapeldateien 152

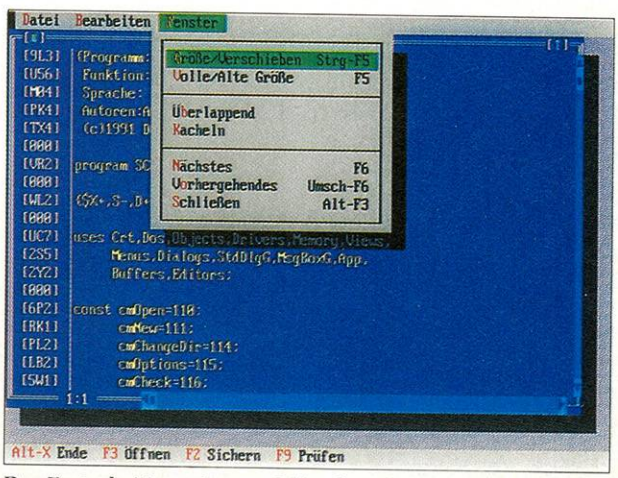
RUBRIKEN

Editorial
Eine Frage der Perspektive 3

Impressum 162



Disketten und andere Massenspeicher gehören zu den wichtigsten Komponenten des PC. Details über ihre Verwaltung durch MS-DOS lesen Sie ab **Seite 106**



Der Fortschritt macht vor SCheck nicht halt. Die neue Version mit der SAA-Oberfläche von Turbo Pascal finden Sie ab **Seite 145**

Assembler-Präprozessor

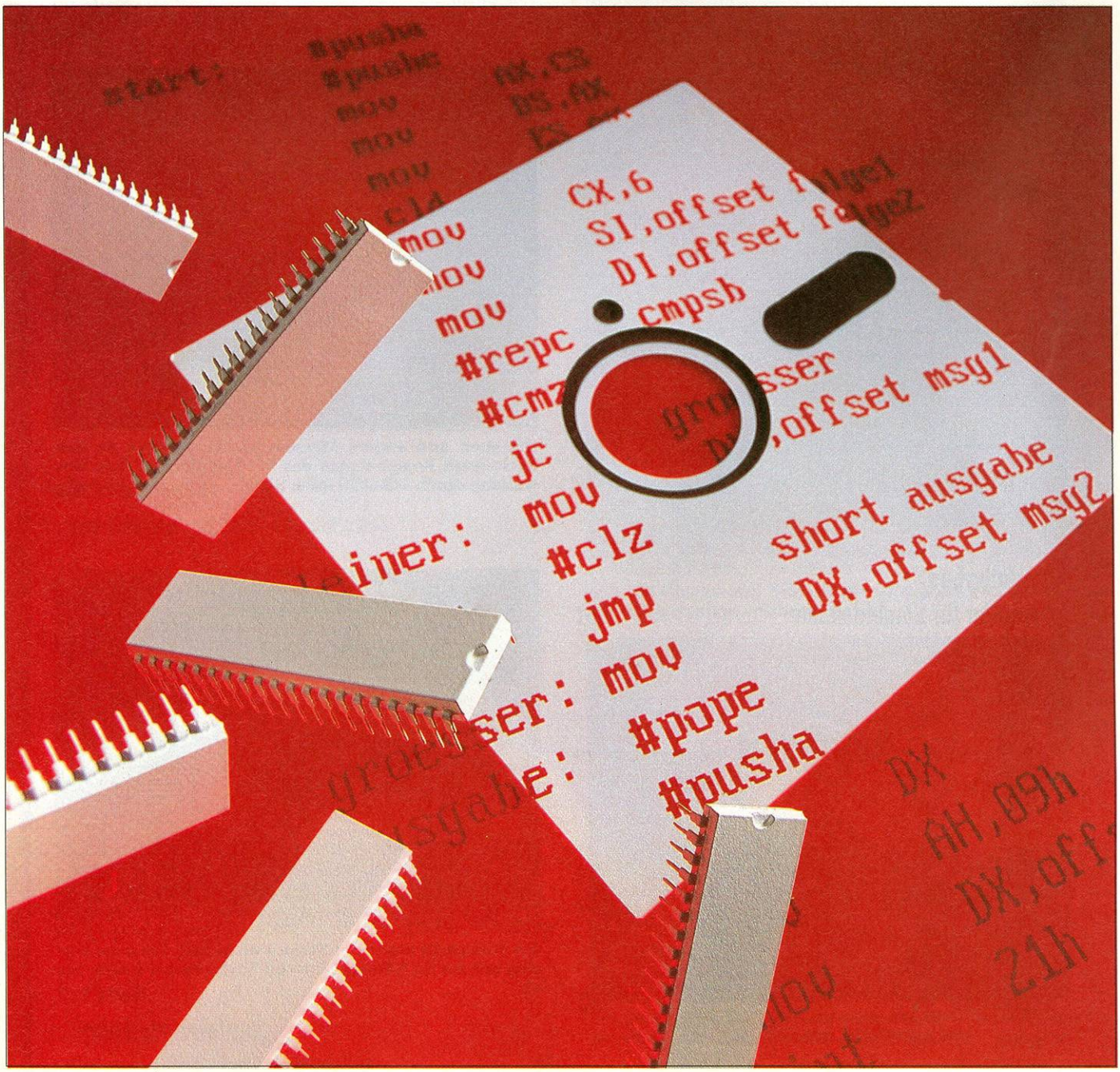
Neue Befehle für den Assembler

Wenn Sie in Assembler programmieren, steht Ihnen immer der gleiche Befehlssatz zur Verfügung, der prozessorspezifisch ist und nur leicht zwischen den Prozessoren 8086, 80286, 80386 und 80486 variiert. Der Assembler-Präprozessor ändert diesen Zustand durch die Bereitstellung neuer Assembler-Direktiven.

Die Erweiterung des Assembler-Befehlssatzes ist eine praktisch unlösbare Aufgabe, da zum Beispiel der Assembler-Befehl

```
mov AX,1234h
```

lediglich eine einfachere Schreibweise der binären Zahlenfolge »10111000001-1010000010010« darstellt. Aus solchen Zahlenfolgen ist die Maschinensprache aufgebaut, die einzige Sprache, die der Prozessor versteht. Er verarbeitet diese binären Zahlenfolgen und leitet entsprechende Aktionen ein, in diesem Fall das Laden des AX-Registers mit dem Wert 1234hex. Die Assembler-Sprache besteht also wie die menschliche Sprache aus Worten und Zahlen und ist deshalb leichter verständlich als abstrakte binäre Zahlen-



folgen. Assembler ist insofern eingeschränkt, als es nur so viele Assembler-Befehle gibt, wie Zahlenäquivalente existieren. Die Erweiterung des Assembler-Befehlssatzes muß deshalb auf einer höheren Ebene erfolgen.

Neben den eigentlichen Assemblern oder Compilern, die ein von Ihnen verfaßtes Programm in Maschinensprache umsetzen, gibt es die sogenannten »Präprozessoren«. Ein Präprozessor ist ein Hilfsprogramm, das einen zu kompilierenden oder zu assemblierenden Quelltext vorbehandelt und für den eigentlichen Compiler beziehungsweise Assembler aufbereitet. Sie kennen diese Einrichtung von der Programmiersprache C her. Definieren Sie zum Beispiel die Konstante »alter« mit dem Wert 25, so lautet die C-Anweisung

```
#define alter 25
```

Das vorangestellte Doppelkreuz »#« signalisiert dem C-Präprozessor, daß der folgende Befehl für ihn und nicht für den Compiler bestimmt ist. Er tauscht im gesamten Quelltext den Bezeichner mit dem Namen »alter« durch den Wert 25 aus und gibt den so »präparierten« Quelltext an den Compiler weiter. Dies ist notwendig, da der Compiler die Anweisung »define« überhaupt nicht kennt.

Der Assembler-Präprozessor »asspre.pas« (Listing 1) arbeitet analog zum oben besprochenen C-Präprozessor. Er »schaltet« sich vor den Assembler, zum Beispiel MASM oder TASM, und ersetzt alle Befehle, die mit einem Doppelkreuz beginnen, durch Assembler-Befehle, die der Assembler und daher auch der Prozessor Ihres PC verstehen. Die neuen Assembler-Befehle stellen also lediglich Synonyme für real existierende Assembler-Befehle dar. Verwechseln Sie diese jedoch nicht mit den Pseudobefehlen des Assemblers. Diese Pseudobefehle, wie zum Beispiel der Befehl »db« (»define byte«), erzeugen im Gegensatz zu den Präprozessorbefehlen keinen eigenen Maschinencode.

Die Bereitstellung neuer Assembler-Befehle kann auch durch Makros erfolgen, die jedoch sehr unflexibel sind und sich durch ihre eingeschränkte Parameterstruktur für dieses Vorhaben nicht eignen. Das Turbo-Pascal-Programm »asspre.pas« analysiert einen eingegebenen Assembler-Quelltext im ASCII-Format und ersetzt alle Befehle, die mit einem Doppelkreuz beginnen, durch ein oder mehrere 80x86-Assembler-Befehle. Der Assembler übersetzt dann in einem zweiten Durchgang diesen präparierten Quelltext in den Maschinencode, den Ihr Rechner ausführt. Tabelle 1 zeigt in Verbindung mit Tabelle 2 die neuen Befehle mit der zugehörigen Syntax. In Tabelle 2 sind alle Bezeichnungen aufgelistet, die in Tabelle 1 Verwendung finden.

Die Multiplikationsbefehle »mulb« (»multiply by byte«) und »mulw« (»multiply by word«) erleichtern Ihnen Multiplikationen von Werten, Registern und Speicherstellen. Die Multiplikation des CX-Registers mit dem Wert 20 in klassischer Assembler-Sprache sehen Sie in Listing 2. Mit dem Präprozessor wird daraus

```
#mulw CX,20
```

Wie Sie Tabelle 1 entnehmen, setzen Sie diese beiden Befehle mit vielen Operandenkombinationen ein. Beachten Sie bitte, daß kein Übertrag in den nächsthöheren Bitbereich erfolgt. Bei einer Multiplikation des AX-Registers (AX = 1000) mit dem Wert 100 erhalten Sie normalerweise das Ergebnis 100000. Der 16-Bit-Bereich ist jedoch nur für Ergebnisse von 0 bis 65535 ausgelegt. Das Ergebnis 100000 benötigt jedoch ein 32 Bit breites Register beziehungsweise ein 16-Bit-Registerpaar. Sie finden in diesem Fall das Ergebnis 34464 im AX-Register vor, das die niederwertigen 16 Bit von 100000 darstellt. Es liegt also in Ihrer Verantwortung, falsche Ergebnisse dieser Art zu vermeiden.

Das Verschieben eines Wertes um mehrere Bit erledigen Sie jetzt mit einem Befehl

Der Assembler-Präprozessor erweitert darüber hinaus die Bit-Rotations- und Schiebebefehle. Mit dem Kommando

```
#shl wert,5
```

schieben Sie den Inhalt der Speicherstelle »wert« um fünf Bit nach links, was in diesem Fall einer Multiplikation der Speicherstelle »wert« mit 32 gleichkommt.

Eine weitere Neuerung stellen die Stringbefehl-Präfixe »repc« und »repnc« dar. Diese Präfixe unterstützen zum Beispiel die Prozessoren NEC V20 und V30, die zum 8088 voll kompatibel sind, aber eben noch ein bißchen mehr können. Das Präfix »repc« (»repeat while carry«) wiederholt die nachgestellte Stringanweisung so lange, bis der Prozessor das Carry-Statusbit löscht. Sehen Sie sich hierzu »beispiel.asp« (Listing 3) an. Es zeigt ein Beispielprogramm zur Verwendung des Repc-Präfix. Das Programm vergleicht zwei Zahlenfolgen und gibt auf dem Bildschirm aus, ob die erste Zahlenfolge größer oder kleiner als die zweite Zahlenfolge ist. Um das Beispielprogramm lauffähig zu machen, geben Sie folgende Befehle ein:

```
asspre beispiel
masm beispiel;
link beispiel;
exe2bin beispiel.exe beispiel.com
```

Für Besitzer des Turbo Assemblers TASM lautet die Sequenz:

```
asspre beispiel
tasm beispiel;
tlink beispiel /t;
```

Um ein Programm durch den Assembler-Präprozessor zu präparieren, geben Sie einfach »asspre« und einen Dateinamen ein. Der Dateiname darf hierbei eine Laufwerksangabe sowie eine komplette Pfadbeschreibung beinhalten. Diese Angaben sind jedoch nicht zwingend. Wenn Sie keinen Dateinamen angeben, gibt Ihnen Asspre eine kurze Hilfestellung auf dem Bildschirm aus. Der Präprozessor setzt für alle Quelltextdateien die Endung »asp« voraus, die Asspre automatisch an den Dateinamen anhängt. Sollten Sie dennoch eine andere Endung angeben, so akzeptiert Asspre diese Variante ebenfalls.

Die drei Kommandos »clz«, »stz« und »cmz« stellen Äquivalente zu den Prozessorbefehlen »clc«, »stc« und »cmc« dar. Mit der Clz-Direktive löschen, mit »stz« setzen und mit »cmz« komplementieren Sie das Zero-Statusbit.

Der neue Befehl »clr« erlaubt Ihnen das Löschen eines Registers beziehungsweise einer Speicherstelle. Durch die Anweisung

```
#clr AX
```

löschen Sie zum Beispiel das AX-Register. Die nun folgenden Befehle gehören zur Gattung der Bitmanipulationsbefehle. Sie erlauben Ihnen, Bits von Registern oder Speicherstellen gezielt zu testen, zu setzen, zu löschen oder zu komplementieren. Um in Erfahrung zu bringen, ob das vierte Bit im DX-Register gesetzt oder gelöscht ist, benutzen Sie den Befehl:

```
#tstb DX,4
```

Der Prozessor setzt das Zero-Statusbit, wenn das vierte Bit im DX-Register gelöscht ist. Der Befehl »setb« setzt das angegebene Bit in einem Register oder in einer Speicherstelle, die Befehle »clrb« und »notb« löschen beziehungsweise komplementieren ein Bit. Die Bitnummer, die der Präprozessor als zweite Angabe erwartet, darf hierbei für 8-Bit-Register oder Speicherstellen im Bereich null bis sieben, für 16-Bit-Register im Bereich null bis 15 liegen.

Der Assembler-Präprozessor stellt zusätzlich neue Stapelbefehle zur Verfügung. Das Kommando »pusha« legt alle Prozessorregister auf den Stapel. Diesen Befehl kennen Sie von den Intel-Prozessoren 80286/386/486. Er legt die Prozessorregister in der Reihenfolge »AX«, »CX«, »DX«, »BX«, »SP«, »BP«, »SI«, »DI« auf dem Stapel ab. Der Pusha-Befehl ist ein Zugeständnis für alle Intel-8086/88-Besitzer. »popa« ist das Gegenstück zum Pusha-Befehl. Er restauriert die oben angegebenen Register, indem er sie in umgekehrter Reihenfolge vom Stapel holt.

Befehl/Syntax	Funktion	Bemerkung
8-Bit-Multiplikation		
mulb reg8,n	reg8 mit Wert n multiplizieren	kein Übertrag im 16-Bit-Bereich
mulb adr8,n	adr8 mit Wert n multiplizieren	kein Übertrag im 16-Bit-Bereich
mulb reg8,reg8	reg8 mit reg8 multiplizieren	kein Übertrag im 16-Bit-Bereich
mulb adr8,reg8	adr8 mit reg8 multiplizieren	kein Übertrag im 16-Bit-Bereich; reg8 darf nicht »AH oder AL« sein
mulb adr8,adr8	adr8 mit adr8 multiplizieren	kein Übertrag im 16-Bit-Bereich
16-Bit-Multiplikation		
mulw reg16,n	reg16 mit Wert n multiplizieren	kein Übertrag im 32-Bit-Bereich
mulw adr16,n	adr16 mit Wert n multiplizieren	kein Übertrag im 32-Bit-Bereich
mulw reg16,reg16	reg16 mit reg16 multiplizieren	kein Übertrag im 32-Bit-Bereich
mulw adr16,reg16	adr16 mit reg16 multiplizieren	kein Übertrag im 32-Bit-Bereich; reg16 darf nicht »AX« sein
mulw adr16,adr16	adr16 mit adr16 multiplizieren	kein Übertrag im 32-Bit-Bereich
Bit-Rotations- und Schiebebefehle		
sal reg,n	alle Bits in reg um n Stellen arithmetisch nach links schieben	
sal reg,reg8	wie oben, jedoch um reg8 Stellen nach links schieben	wenn reg=»CH/CL/CX«, dann reg8<>»AH,AL«
sal reg,adr8	wie oben, jedoch um Anzahl Stellen in adr8 nach links schieben	
sal adr,n	siehe sal reg,n	
sal adr,reg8	siehe sal reg,reg8	
sal adr,adr8	siehe sal reg,adr8	
shl op1,op2	siehe alle Sal-Befehle, jedoch logisch links schieben	siehe alle Sal-Befehle
sar op1,op2	siehe alle Sal-Befehle, jedoch arithmetisch rechts schieben	siehe alle Sal-Befehle
shr op1,op2	siehe alle Sal-Befehle, jedoch logisch rechts schieben	siehe alle Sal-Befehle
rcl op1,op2	siehe alle Sal-Befehle, jedoch links durch Carry-Flag rotieren	siehe alle Sal-Befehle
rcr op1,op2	siehe alle Sal-Befehle, jedoch rechts durch Carry-Flag rotieren	siehe alle Sal-Befehle
rol op1,op2	siehe alle Sal-Befehle, jedoch links rotieren	siehe alle Sal-Befehle
ror op1,op2	siehe alle Sal-Befehle, jedoch rechts rotieren	siehe alle Sal-Befehle
Stringbefehl-Präfixe		
repc strbef	Befehlspräfix für Stringbefehle (»repeat while carry«)	nur sinnvoll bei scas/cmps
repnc strbef	Befehlspräfix für Stringbefehle (»repeat while not carry«)	nur sinnvoll bei scas/cmps
Statusbit-beeinflussende Befehle		
clz	Zero-Flag löschen	
stz	Zero-Flag setzen	
cmz	Zero-Flag komplementieren	
Bitmanipulationsbefehle		
tstb reg8,b8	Bit b8 in reg8 testen	Statusbits werden gemäß Prozessorbefehl »test« gesetzt
tstb reg16,b16	Bit b16 in reg16 testen	siehe tstb reg8,b8
tstb adr8,b8	Bit b8 an Adresse adr8 testen	siehe tstb reg8,b8
tstb adr16,b16	Bit b16 an Adresse adr16 testen	siehe tstb reg8,b8
setb	siehe alle Tstb-Befehle	Bit setzen
clrb	siehe alle Tstb-Befehle	Bit löschen
notb	siehe alle Tstb-Befehle	Bit komplementieren
Stapelbefehle		
pusha	alle Prozessorregister auf den Stapel legen	
popa	alle Prozessorregister vom Stapel holen	
pushe	alle erweiterten Register auf den Stapel legen	
pope	alle erweiterten Register vom Stapel holen	
pushv nn	Wert nn auf den Stapel legen	
Diverse Befehle		
clr reg	Register reg löschen	
clr adr	Speicherstelle adr löschen	

Tabelle 1. Die neuen Assembler-Befehle im Überblick

Eine weitere Neuerung ist der Pushv-Befehl (»push value«). Die Pushv-Anweisung legt einen 16-Bit-Wert direkt auf dem Stapel ab:

```
#pushv 1234h
```

In diesem Beispiel liegt dann der Wert 1234hex an erster Stelle auf dem Stapel. Sie verwenden »pushv« zum Beispiel zum direkten Ablegen von Parametern für aufzurufende Unterprogramme. Dieser Befehl entstammt wiederum dem NEC-V20/V30-Prozessor.

Acht neue Universalregister erleichtern Ihnen das Programmieren

Der Assembler-Präprozessor stellt Ihnen jedoch nicht nur neue Befehle zur Verfügung, sondern erweitert Ihren Prozessor sogar um einige Eigenschaften. Hierzu zählt die Bereitstellung von acht neuen Prozessorregistern. Asspre emuliert die neuen 16-Bit-Register »EX«, »FX«, »GX«, »IX«, »KX«, »MX«, »NX«, »PX«, die Sie jeweils in zwei 8-Bit-Register (»EH«, »EL«, »FH«, »FL« etc.) aufteilen können. Das herausragende Merkmal der neuen Register (im folgenden als »erweiterte Register« bezeichnet) ist, daß Sie diese uneingeschränkt wie die »normalen« Prozessorregister einsetzen können. Einige Beispiele verdeutlichen dies:

```
#mov AX, EX
#add PH, FL
#mov <Speicherstelle>,MX
#push GX
```

Beachten Sie hierbei bitte, daß Sie allen Befehlen, die die erweiterten Register benutzen, ein Doppelkreuz voranstellen. Die beiden neuen Kommandos »pushe« (»push all extended registers«) und »pope« (»pop all extended registers«) arbeiten analog zu den beiden Befehlen »pusha« und »popa« mit dem Unterschied, daß sie sich auf die erweiterten Register beziehen. Herausragend ist zudem, daß auch alle erweiterten Register mit den neuen Befehlen aus Tabelle 1 zusammenarbeiten. Womöglich fragen Sie sich, warum die Bezeichnungen der erweiterten Register nicht alphabetisch fortlaufend sind. Dies hat folgenden Grund: Ein Register mit der Bezeichnung »HX« könnten Sie in die beiden 8-Bit-Register »HH« und »HL« aufteilen. Die Bezeichnung »HH« ist jedoch nicht so leicht als Register identifizierbar. Der Präprozessor emuliert deshalb nur solche Register, deren 8-Bit-Hälften durch zwei unterschiedliche Buchstaben gekennzeichnet werden.

Ein weiteres Leistungsmerkmal des Asspre ist die Option, die Ausführung jedes Assembler-Befehls vom Zustand eines Statusbits abhängig zu machen. Sie müssen hierzu dem Befehl das Doppelkreuz voranstellen und die Be-

zeichnung eines Statusbits mit einem Unterstrich hinter den Assembler-Befehl schreiben. Die Syntax dieser Einrichtung lautet:

```
#<Assemblerbefehl>_cc [op1][,][op2]
```

Die Buchstaben »cc« stehen hierbei für »condition code« (deutsch: »Bedingungscode«). Der Präprozessor erlaubt für »cc« alle Statusbitangaben, die auch beim Assemblerbefehl »jmp« erlaubt sind, zum Beispiel:

```
#mov_nz AX,1
#jmp_ae 1234h:4567h
#retf_cxz
```

Der Prozessor führt zum Beispiel den Mov-Befehl nur aus, wenn das Zero-Statusbit gelöscht ist. Für diese Neuerung gilt wiederum, daß Sie sie auch bei allen neuen Befehlen und mit den erweiterten Registern einsetzen können:

```
#shl_be GX,10
```

Ursprünglich war diese Einrichtung für die Befehle »call«, »jmp« und »ret« der Intel-Prozessoren implementiert, da zum Beispiel bedingte Sprünge nur im Short-Bereich (128 Speicherstellen rückwärts oder 127 Speicherstellen vorwärts) erlaubt sind. Der Präprozessor spaltet von diesen Kombinationen die Angaben nach dem Unterstrich, also den eigentlichen Bedingungscode, ab, und setzt ihn hinter ein »j«. Dadurch entsteht ein normaler bedingter Sprung, der gegebenenfalls hinter die fragliche Anweisung, die als nächstes kommt, verzweigt.

Der Assembler-Präprozessor bietet Ihnen die zusätzliche Variante, daß Sie beide Operanden eines Assembler-Befehls als Speicherstellen angeben, zum Beispiel:

```
#add wert1,wert2
```

Der Präprozessor benötigt jedoch bei mindestens einem der beiden Operanden eine Bereichsangabe, so daß die korrekte Anweisung lautet:

```
#add byte ptr wert1,wert2
```

Diese Schreibweise gilt, wenn die beiden Variablen »wert1« und »wert2« vom Typ »byte« sind. Achten Sie bitte darauf, daß zwischen der Bereichsangabe »byte« und der Anweisung »ptr« nur ein Leerzeichen stehen darf.

Es versteht sich natürlich von selbst, daß auch Zeilen, die Sie durch das Doppelkreuz für den Präprozessor bestimmen, die Komponenten Label, Befehl, Operanden und Bemerkung beinhalten dürfen, zum Beispiel:

```
loop: #add EX,2 ;Bemerkung
```

Bei der Arbeit mit Asspre müssen Sie noch folgende Dinge beachten: Der Assembler-Präprozessor ist sehr tolerant, das bedeutet, es erfolgt keine Überprüfung der Operanden, die Sie im Zusammenhang mit Assembler-Befehlen angeben. Dies hat den Vorteil, daß der Präprozessor sehr kurz ist, aber dennoch über ein großes Befehlsvolumen verfügt. Bevor Sie die neuen Assembler-Befehle in Ihrem Programm benutzen, müssen Sie dessen Stapel initialisieren. Die ersten 18 Byte des Stapels benötigt Asspre für die erweiterten Register und zum Zwischenspeichern von Prozessorregistern. Definieren Sie deshalb den Stapel lieber größer, damit Ihr Programm nicht die ersten 18 Byte des Stapels beschreibt. Die Listings 4 und 5 zeigen die generellen Strukturen von Com- und Exe-Programmen, die Sie für die Zusammenarbeit mit dem Präprozessor benötigen. Asspre beansprucht für seine Arbeit intern einige Sprungmarken, die Sie als reserviert ansehen müssen und in Ihrem Programm natürlich nicht verwenden dürfen. Der Präprozessor benutzt die Label »LAnnnnn«, »LBnnnnn«, »RAnnnnn« und »RBnnnnn«, wobei die Buchstaben »nnnnn« für eine Zahl zwischen null und 65535 stehen.

Edtnum korrigiert die verschobenen Zeilennummern von Fehlermeldungen

Da der Präprozessor für einige neue Befehle aus einer Quelltextzeile mehrere Zeilen erzeugt, stimmen die Zeilenangaben, die der Assembler im Zusammenhang mit fehlerhaften Zeilen ausgibt, nicht mehr mit den Zeilennummern des Quelltextes überein. Dies erschwert das Debuggen eines Assembler-Programms erheblich, da Sie die fehlerhafte Zeile erst im Quelltext suchen müssen. Das Hilfsprogramm »edtnum.pas« (Listing 6) korrigiert diese falschen Zeilennummern. Der Präprozessor erzeugt eine Datei mit der Erweiterung »num«, die Informationen zum Korrigieren der Zeilennummern beinhaltet. Die Stapeldatei »ass.bat« (Listing 7), die den gesamten Assembliervorgang automatisiert, leitet hierzu alle Ausgaben des Assemblers in eine Datei um. Edtnum analysiert diese Datei, berechnet gefundene Zeilennummern neu und trägt die richtigen Zeilennummern in eine Zieldatei ein. Der Inhalt dieser Zieldatei erscheint dann auf dem Bildschirm. Vergewissern Sie sich, daß sich der Präprozessor, der Assembler, der Linker und die Datei »edtnum.exe« im selben Verzeichnis befinden. Das Programm »edtnum« arbeitet jedoch nur mit den Assemblern MASM ab Version 5.1 oder TASM beliebiger Version zusammen. Die Ausgaben des MASM 4.0 lassen sich zwar auch in eine Datei umleiten; die Fehlermeldungen gibt dieser jedoch immer auf dem Bildschirm aus.

Sollten Sie das Hilfsprogramm Edtnum nicht verwenden, so können Sie auch direkt die vom Präprozessor erzeugte Asm-Datei bearbeiten, für die die Zeilennummern stimmen. Der Präprozessor erleichtert Ihnen hierbei das Differenzieren zwischen Präprozessordirektiven und »normalen« Assembler-Befehlen durch das Markieren der Befehlssequenzen, die von ihm stammen. Er fügt unmittelbar vor und nach der eingefügten Sequenz eine Kommentarzeile ein, die einen Verweis auf den ursprünglichen Präprozessorbefehl enthält. Danach assemblieren und linken Sie die so verbesserte Asm-Datei, ohne den Präprozessor erneut zu benutzen. Dieser braucht den Quelltext für den Assembler nur einmal zu präparieren.

Die zweite Variante bietet darüber hinaus den Vorteil, daß Sie auch Quelltextzeilen verbessern können, die innerhalb einer Befehlssequenz des Präprozessors liegen. Dieser Zustand tritt in Erscheinung, wenn Sie zum Beispiel bei Präprozessorbefehlen Register angeben, die nicht existieren. Fehler dieser Art korrigiert der Präprozessor allerdings nicht.

Sie erfahren nun anhand der Befehle »tstb«, »setb«, »clrb« und »notb«, wie der Präprozessor intern arbeitet. Den Befehl

```
#tstb AH,6
```

übersetzt der Präprozessor in

Bezeichnung	Erläuterung
reg8	gültiges 8-Bit-Register
reg16	gültiges 16-Bit-Register
adr8	Adresse einer 8-Bit-Speicherstelle
adr16	Adresse einer 16-Bit-Speicherstelle
reg	gültiges Register (8 oder 16 Bit)
adr	Adresse einer Speicherstelle (8 oder 16 Bit)
n	8-Bit-Wert
nn	16-Bit-Wert
strbef	Stringbefehl (zum Beispiel movs, stos, lods, cmps, scas)
op1	Operand 1
op2	Operand 2
b8	Bitnummer im Bytebereich (0 bis 7)
b16	Bitnummer im Wortbereich (0 bis 15)

Tabelle 2. Die in Tabelle 1 verwendeten Bezeichnungen

```
test AH,1 shl 6
```

Der Assembler ersetzt den Operanden »1 shl 6« durch den Wert 64. So erhält er den Befehl:

```
test AH,64
```

Dieser Befehl ist ein normaler 80x86-Assembler-Befehl, den der Assembler schließlich in den äquivalenten Maschinencode übersetzt. Der Präprozessor simuliert die Befehle »setb«, »clrb« und »notb« durch die Assembler-Befehle »or«, »and« und »xor«. Die erweiterten Register ersetzt dieser durch Speicherstellenbezüge. Den Befehl

```
#mov EX,1234h
```

übersetzt er zu

```
mov word ptr SS:[2],1234h
```

Deshalb müssen Sie auch die ersten 18 Byte des Stapels reservieren, da dort die Werte der erweiterten Register gespeichert sind. Die Anweisung

```
#mov EX,FX
```

würde der Präprozessor demnach in den Assembler-Befehl

```
mov word ptr SS:[2],word ptr SS:[4]
```

übersetzen. Diesen Sonderfall erkennt »asspre« und erzeugt einen modifizierten Quellcode, der das AX-Register zum Zwischenspeichern verwendet. Bei allen Befehlen gilt übrigens, daß die Registerinhalte aller unbeteiligten Register nicht verändert werden.

Die Programmstruktur des Präprozessors läßt Erweiterungen jeder Art zu. Um zum Beispiel noch weitere Befehle zu implementieren, müssen Sie die Bezeichnung des Befehls in das Feld »mne« eintragen und die Konstante »maxmne« erhöhen, die die Anzahl der verfügbaren Befehle wiedergibt. Danach integrieren Sie in die Case-Anweisung der Prozedur »analyse« einen weiteren Programmabschnitt. Die beiden Variablen »op1str« und »op2str« beinhalten gegebenenfalls die beiden Operanden, die Ihrem neuen Befehl als Großbuchstaben übergeben wurden. Alle Assembler-Befehle, durch die der Präprozessor die ursprüngliche Direktive ersetzen soll, schreiben Sie in den Befehlsspeicher »bef«. Der Befehlsspeicher ist ein Feld vom Typ »String«, das maximal 20 Einträge aufnehmen kann. Zum Schluß setzen Sie die Variable »befcnt« auf die Anzahl der Befehle im Befehlsspeicher. Asspre übernimmt dann alle weiteren Aufgaben für Sie. Darunter fällt das Einfügen der korrekten Operanden in die Assembleranweisung ebenso wie das Ersetzen der entsprechenden Zeilen in der Ausgabedatei.

Die Assembler-Programmierung ist bekanntermaßen eine schweißtreibende Angelegenheit, die Ihnen Ihr ganzes Können abfordert. Gerade deshalb werden Sie die Arbeitserleichterung, die Ihnen der Präprozessor verschafft, bald zu schätzen wissen.

(Ingo Kubbilun/wn)

```
[UV3]    1: { Programm: asspre.pas
[KA5]    2: Funktion: Bereitstellung neuer
[EK3]    3: Assembler-Direktiven
[ZS5]    4: Spache   : Turbo Pascal ab Version 4.Ø
[ØR3]    5: Autor    : Ingo Kubbilun
[485]    6: (C) 1991 DMV Widuch GmbH & Co. KG }
[ØØØ]    7:
[ØØØ]    8:
[DW5]    9: program Assembler_Preprocessor;
[ØØØ]   1Ø:
[5H1]   11: uses Crt;
[ØØØ]   12:
[VD6]   13: const reg8: array[Ø..15] of string[2]=
[BN2]   14:              ('EL','EH','FL','FH',
[SL2]   15:               'GL','GH','IL','IH',
[ØN2]   16:               'KL','KH','ML','MH',
[HT2]   17:               'NL','NH','PL','PH');
[T45]   18:       reg16: array[Ø..7] of string[2]=
[7U2]   19:            ('EX','FX','GX','IX',
[AX2]   2Ø:             'KX','MX','NX','PX');
[MN1]   21:       maxmne=24;
[354]   22:       mne: array[Ø..maxmne] of
[RX1]   23:       string[6]=
[V23]   24:         ('MULB','MULW','SAL',
[ZH2]   25:          'SHL','SAR','SHR',
[YH2]   26:          'RCL','RCR','ROL',
[E53]   27:          'ROR','REPC','REPNC',
[RK2]   28:          'CLZ','STZ','CMZ',
[ØY2]   29:          'CLR','TSTB','SETB',
[S83]   3Ø:          'CLRB','NOTB','PUSHA',
[5C3]   31:          'POPA','PUSHE','POPE',
[KD1]   32:          'PUSHV');
[ØØØ]   33:
[TM1]   34: var file1,
[4YØ]   35:     file2,
[6YØ]   36:     file3,
[CA1]   37:     linstr,
[481]   38:     labstr,
[8A1]   39:     remstr,
[T91]   4Ø:     mnestr,
[L71]   41:     op1str,
[ZP2]   42:     op2str: string;
[TE3]   43:     source,destin: Text;
[ML3]   44:     number: file of Byte;
[EE3]   45:     ncnt,p,znr: Integer;
[1X1]   46:     zcnt: Byte;
[S93]   47:     repcnt,labcnt: Word;
[872]   48:     befcnt: Byte;
[673]   49:     bef: array[1..2Ø] of
[GC1]   5Ø:     string;
[GX1]   51:     opnr: Byte;
[ØØØ]   52:
[ØØØ]   53:
[NG5]   54: procedure UpStr (var s: string);
[UR6]   55: { wandelt alle Buchstaben eines Strings
[CC4]   56: in Großbuchstaben um }
[ØØØ]   57:
[ZH2]   58: var i: Integer;
[A82]   59:     flag: Boolean;
[ØØØ]   6Ø:
[MUØ]   61: begin
[ZØ2]   62:   flag:=False;
[8U3]   63:   for i:=1 to Length(s) do
[MUØ]   64:   begin
```

```
[EW3]   65:     flag:=flag Xor (s[i]='"');
[ØV5]   66:     if NOT flag then s[i]:=UpCase(s[i])
[4MØ]   67:   end
[ZNØ]   68: end;
[ØØØ]   69:
[BL5]   7Ø: procedure IgnSpc (var s: string);
[6G7]   71: { überließt alle Leerzeichen bzw. Tabu-
[126]   72: latoren, die am Anfang eines Strings
[8Z2]   73: gefunden werden }
[ØØØ]   74:
[MUØ]   75: begin
[Z65]   76:   while (Length(s)<>Ø) AND ((s[1]=' ')
[N82]   77:   OR (s[1]=#9)) do
[MU1]   78:   Delete(s,1,1)
[ZNØ]   79: end;
[ØØØ]   8Ø:
[DL5]   81: procedure CutSpc (var s: string);
[LR6]   82: { schneidet alle Leerzeichen bzw. Tabu-
[A36]   83: latoren ab, die am Ende eines Strings
[8Z2]   84: gefunden werden }
[ØØØ]   85:
[DK2]   86: var p: Integer;
[A82]   87:     flag: Boolean;
[ØØØ]   88:
[MUØ]   89: begin
[532]   9Ø:   p:=Length(s);
[ZØ2]   91:   flag:=False;
[S94]   92:   while (p>Ø) AND NOT flag do
[MUØ]   93:   begin
[8U1]   94:     if (s[p]=' ')
[6U4]   95:     OR (s[p]=#9) then Delete(s,p,1)
[SR2]   96:     else flag:=True;
[6XØ]   97:     Dec(p)
[4MØ]   98:   end
[ZNØ]   99: end;
[ØØØ]  1ØØ:
[LT5]  1Ø1: function XPos (ch: Char; s: string)
[LL1]  1Ø2: : Integer;
[SV6]  1Ø3: { sucht ein Zeichen in einem String und
[5N5]  1Ø4: gibt die Position des Zeichens als
[2T5]  1Ø5: Ergebnis zurück. Das Zeichen wird
[9Y5]  1Ø6: ignoriert, sobald es in Anführungs-
[ZØ3]  1Ø7: strichen steht. }
[ØØØ]  1Ø8:
[P34]  1Ø9: var flag,found: Boolean;
[PV1]  11Ø:     p: Integer;
[ØØØ]  111:
[MUØ]  112: begin
[ZØ2]  113:   flag:=False;
[S72]  114:   found:=False;
[9RØ]  115:   p:=1;
[TT5]  116:   while (p<=Length(s)) AND NOT found do
[MUØ]  117:   begin
[AX3]  118:     flag:=flag Xor (s[p]='"');
[D95]  119:     if NOT flag then found:=s[p]=ch;
[BYØ]  12Ø:     Inc(p)
[ZNØ]  121:   end;
[RT3]  122:   if found then XPos:=p-1
[E22]  123:   else XPos:=Ø
[ZNØ]  124: end;
[ØØØ]  125:
[VG5]  126: procedure ChgReg (var s: string);
[BY5]  127: { tauscht ein erweitertes Register
[WL5]  128: ex,fx,gx,ix,kx,mx,nx oder px durch
```

```
[6F5]  129: einen Speicherstellenbezug aus }
[ØØØ]  13Ø:
[ZH2]  131: var i: Integer;
[Y32]  132:     adr: string;
[BL2]  133:     found: Boolean;
[ØØØ]  134:
[MUØ]  135: begin
[UPØ]  136:   i:=Ø;
[S72]  137:   found:=False;
[VB4]  138:   while (i<16) AND NOT found do
[MUØ]  139:   begin
[RW2]  14Ø:     found:=reg8[i]=s;
[XXØ]  141:     Inc(i)
[ZNØ]  142:   end;
[Ø42]  143:   if found then
[MUØ]  144:   begin
[A22]  145:     Str(i+1,adr);
[W54]  146:     s:='BYTE PTR SS:['+adr+']'
[4MØ]  147:   end
[ZRØ]  148:   else
[MUØ]  149:   begin
[UPØ]  15Ø:     i:=Ø;
[S72]  151:     found:=False;
[4D4]  152:     while (i<8) AND NOT found do
[MUØ]  153:     begin
[943]  154:       found:=reg16[i]=s;
[XXØ]  155:       Inc(i)
[ZNØ]  156:     end;
[Ø42]  157:     if found then
[MUØ]  158:     begin
[ØT2]  159:       Str((i-1)*2+2,adr);
[Z54]  16Ø:       s:='WORD PTR SS:['+adr+']'
[4MØ]  161:     end
[4MØ]  162:   end
[ZNØ]  163: end;
[ØØØ]  164:
[MS5]  165: function GetMne (s: string): Byte;
[XX6]  166: { gibt die Nummer eines Befehls (Mnemonic)
[XH6]  167: als Ergebnis zurück, Ø bedeutet Befehl
[W52]  168: unbekannt }
[ØØØ]  169:
[ZH2]  17Ø: var i: Integer;
[BL2]  171:     found: Boolean;
[ØØØ]  172:
[MUØ]  173: begin
[UPØ]  174:   i:=Ø;
[S72]  175:   found:=False;
[WE5]  176:   while (i<maxmne+1) AND NOT found do
[MUØ]  177:   begin
[PX2]  178:     found:=s=mne[i];
[951]  179:     Inc(i);
[ZNØ]  18Ø:   end;
[WW3]  181:   if found then GetMne:=i
[4K2]  182:   else GetMne:=Ø;
[ZNØ]  183: end;
[ØØØ]  184:
[616]  185: function TstOp (s: string): Boolean;
[KZ6]  186: { liefert True, wenn Operand ein Register
[XE4]  187: oder eine Zahl darstellt }
[ØØØ]  188:
[RD6]  189: const reg8: array [Ø..7] of string[2]=
[BM2]  19Ø:       ('AH','AL','BH','BL',
[SN2]  191:       'CH','CL','DH','DL');
[S65]  192:       reg16: array[Ø..11] of string[2]=
[WS2]  193:       ('AX','BX','CX','DX',
[KN2]  194:       'SI','DI','BP','SP',
[TT2]  195:       'CS','DS','ES','SS');
[ØØØ]  196:
[HS3]  197: var i,z,zerr: Integer;
[BL2]  198:     found: Boolean;
[R92]  199:     zstr: string;
[ØØØ]  2ØØ:
[MUØ]  2Ø1: begin
[UPØ]  2Ø2:   i:=Ø;
[S72]  2Ø3:   found:=False;
[4D4]  2Ø4:   while (i<8) AND NOT found do
[MUØ]  2Ø5:   begin
[VY2]  2Ø6:     found:=s=reg8[i];
[XXØ]  2Ø7:     Inc(i)
[ZNØ]  2Ø8:   end;
[8R2]  2Ø9:   if NOT found then
[MUØ]  21Ø:   begin
[UPØ]  211:     i:=Ø;
[BB4]  212:     while (i<12) AND NOT found do
[MUØ]  213:     begin
[743]  214:       found:=s=reg16[i];
[XXØ]  215:       Inc(i)
[4MØ]  216:     end
[ZNØ]  217:   end;
[8R2]  218:   if NOT found then
[MUØ]  219:   begin
[642]  22Ø:     zstr:=s[1];
[DZ2]  221:     Val(zstr,z,zerr);
[482]  222:     found:=zerr=Ø
[ZNØ]  223:   end;
[172]  224:   TstOp:=found
[ZNØ]  225: end;
[ØØØ]  226:
[116]  227: function TstBW (s: string): Boolean;
```

Listing 1. Der Assembler-Präprozessor »asspre.pas« verhilft dem Assembler zu neuen Fähigkeiten

BATH BEACH
SOFTWARE-SERVICE

PUBLIC DOMAIN 150

Translator übersetzt englische Programm-Dokumentationen in Deutsch, **Lightning-Press** DTP-Desktop-Publishing, Fantastic **As Easy As** sehr gutes Tabellenkalkulationsprogramm, Bestseller **Prophet-Lotto** Statistikprogramm für Lotterieziehungen, **3D-Chess** dreidimensionales Schachprogramm, **PC-Write** sehr gute Textverarbeitung mit vielen Möglichkeiten, **Free-File** sehr gute Datenverwaltung mit vielen Möglichkeiten, **Composer** Musik komponieren, Noten schreiben und ausspielen, **PC-Akt** erzeugt digitalisierte Aktbilder auf ihrem PC, **Dancad-3D** dreidimensionales CAD-Zeichenprogramm mit Animation, **PC-KWIK** beschleunigt ihr Diskettenlaufwerk, **Mahjongg** sehr gutes chinesisches Spiel mit guter Grafik **Megagopoly** Spiel, wie erreiche ich eine Million, **Nuke New York** zeigt einen nuklearen Angriff und die Auswirkungen auf New York **Designer** ein Zeichenprogramm, das bewegliche Bilder erstellt **Black Jack** spielen wie im Casino **Horoskop** erstellt Ihr persönliches Horoskop **Kniffel** Kartenspiel **Mastermind** populäres Strategiespiel **Star-Traders** Händlerspiel, Aktien kaufen im Weltraum **Etikett** erstellt Etiketten in verschiedenen Formaten und druckt sie aus **Flugsimulator** simuliert Jet-Flüge **PC-Calc** populäre Tabellenkalkulation **Copy-All** nun können Sie fast alle geschützten Programme kopieren **PC-Window** PC-Fenstertechnik **Kalender** erstellt einen Kalender mit Bildschirm/Druck **Printum** schaltet den Drucker in andere Schriftarten um **Bastools** Basic-Tools **Your Menu** erstellt Ihr eigenes Menü zum Selbst-Ausführen **Form-Master** umfangreicher Formular-Generator **DOS-Help** zeigt und erklärt alle DOS-Befehle**StockCharting** Aktienverwaltung,Kauf-/Verkaufssignal(graf. Darstellung) **Biorythm** Biorythmen **Finance** Finanzen und Berechnungen**Taskplan** Zeitplanungsprogramm **Cover** druckt Diskettenhüllen mit Inhaltsverzeichnis **VCR-D-Base** verwaltet Video-Casetten und druckt Aufkleber **Bookinventory** Inventurprogramm für bis zu 2000 Bücher

150 Programme für nur 130.- DM!

P C -Lock schützt ihre Festplatte mit Passwort **Best Utilities** jede Menge Hilfsprogramme **Cataloging** erstellt einen Diskettenkatalog **FB-Copy** ein sehr schnelles und komfortabeles Kopierprogramm **Virus Check** Computerviren-Untersuchungsprogramm **Backgammon** Backgammonspiel **Space-Invaders** sehr schönes, bekanntes Arcade-Spiel **PC-Foto** Fotoverwaltungsprogramm incl. Druck von Dia-Aufklebern **Packdisk** macht mehr Platz auf Ihrer Festplatte **Compress** Datenkompakter, der Platz auf der Diskette spart **Chasm** Super Assembler-Programmiersprache **plus weitere 101 Public Domain- und Shareware-Programme.** *Hilfs-, Anwendungs- und Spielprogramme (BAS-, COM- und EXE-Startdateien) für IBM PC/XT/AT mit VGA, EGA-, CGA- und Herculeskarte.Alle Programme wurden auf Viren geprüft.*

Schnellversand am Tag des Bestelleingangs. Großer INFO-Katalog mit Programm-Beschreibungen und zusätzlichen Angeboten liegt jeder Bestellung bei.

Inklusive deutscher Anleitung zum Starten aller Programme!

© LIESSFELD & Partner LANDAU

Bath Beach
Software-Service GmbH
Speckstraße 23, D-6749 Schweighofen
BTX: * BATH # • FAX:(06342)6216 • Tel: (06343) 5818

BESTELL-COUPON

Ja, bitte senden Sie mir umgehend das PC-Paket "Public Domain 150" zum Discount-Preis von 130.-DM. **Bei Vorkasse keine Versandkosten.**
Bei Nachnahme: DM 10.- Versandkosten. Den Gesamtbetrag (Endpreis) bezahle ich wie angekreuzt. (Ausland nur Vorkasse: Scheck oder Bar).
Zusätzlich erhalte ich **kostenlos** den ausführlichen INFO-Katalog.

Endpreis: _____ ❑ 5¼"Disketten ❑ Scheck/Bar DM 130.- ❑ Per Nachnahme DM 140.-
Endpreis: _____ ❑ 3.5"Disketten ❑ Scheck/Bar DM 150.- ❑ Per Nachnahme DM 160.-
(3.5"Zoll-Disketten Material-Mehrpreis : 20.-)

Name:
Straße: Computer-Typ:
PLZ/ORT. Grafik-Karte:
Telefon:
Datum /Unterschrift
(Bei Minderjährigen der Erziehungsberechtigte) Geburtsdatum DOEX 16 '91

```
[W86]  228: { liefert True, wenn Operand im Wort-
[5U2]  229: bereich liegt }
[MUØ]  23Ø: begin
[YM4]  231:   TstBW:=Pos('WORD PTR',s)<>Ø;
[ZNØ]  232: end;
[ØØØ]  233:
[B45]  234: procedure PutShift (nr: Byte);
[3C7]  235: { erzeugt eine Assemblersequenz für einen
[UK6]  236: eingegebenen Bitmanipulationsbefehl }
[ØØØ]  237:
[ØH2]  238: var achr: Char;
[ØØØ]  239:
[MUØ]  24Ø: begin
[KØ3]  241:   if (op2str='CL') OR
[TN2]  242:   (op2str='1') then
[MUØ]  243:   begin
[Y91]  244:     opnr:=Ø;
[ASØ]  245:     Exit
[ZNØ]  246:   end;
[3B5]  247:   if (op1str<>'CX') AND (op1str<>'CH')
[DR3]  248:   AND (op1str<>'CL') then
[MUØ]  249:   begin
[WW2]  25Ø:     bef[1]:='PUSH CX';
[234]  251:     bef[2]:='MOV CL,'+op2str;
[585]  252:     bef[3]:=mne[nr]+' '+op1str+',CL';
[UT2]  253:     bef[4]:='POP CX';
[4M1]  254:     befcnt:=4;
[4MØ]  255:   end
[ZRØ]  256:   else
[MUØ]  257:   begin
[U84]  258:     if op1str='CX' then achr:='X'
[WD2]  259:     else achr:='L';
[MW2]  26Ø:     bef[1]:='PUSH AX';
[F15]  261:     bef[2]:='MOV A'+achr+','+op1str;
[434]  262:     bef[3]:='MOV CL,'+op2str;
[2Ø5]  263:     bef[4]:=mne[nr]+' A'+achr+',CL';
[8V4]  264:     bef[5]:='MOV '+op1str+',A'+achr;
[ST2]  265:     bef[6]:='POP AX';
[8M1]  266:     befcnt:=6;
[ZNØ]  267:   end;
[ZNØ]  268: end;
[ØØØ]  269:
[WM5]  27Ø: procedure AnalyseLine (s: string);
[R47]  271: { analysiert eine Quelltextzeile und zer-
[DF6]  272: legt diese in Label, Mnemonic, Operanden
[TW2]  273: und Bemerkung }
[FU2]  274: var l,p: Integer;
[4V2]  275:     sreg: string[2];
[ØØØ]  276:
[MUØ]  277: begin
[1K1]  278:   UpStr(s);
[HK1]  279:   IgnSpc(s);
[7M1]  28Ø:   CutSpc(s);
[FB2]  281:   p:=XPos(';',s);
[8T1]  282:   if p<>Ø then
[MUØ]  283:   begin
[4M2]  284:     l:=Length(s)-p+1;
[D93]  285:     remstr:=Copy(s,p,l);
[X32]  286:     Delete(s,p,l);
[PE1]  287:     CutSpc(s)
[ZNØ]  288:   end;
[CB2]  289:   p:=XPos(':',s);
[8T1]  29Ø:   if p<>Ø then
[MUØ]  291:   begin
[ZE3]  292:     labstr:=Copy(s,1,p-1);
[5Ø3]  293:     l:=Length(labstr);
[VA1]  294:     sreg:='';
[DT1]  295:     if l>=2 then
[MUØ]  296:     begin
[C33]  297:       sreg:=labstr[l-1]+
[RV1]  298:       labstr[l];
[DC2]  299:       if (sreg='CS') OR
[E32]  3ØØ:       (sreg='DS') OR
[F32]  3Ø1:       (sreg='ES') OR
[9P1]  3Ø2:       (sreg='SS')
[VF2]  3Ø3:       then labstr:=''
[2V2]  3Ø4:       else Delete(s,1,p)
[ZNØ]  3Ø5:     end;
[ØD1]  3Ø6:     IgnSpc(s)
[ZNØ]  3Ø7:   end;
[Y82]  3Ø8:   p:=XPos(' ',s);
[X83]  3Ø9:   if p=Ø then mnestr:=s
[ZRØ]  31Ø:   else
[MUØ]  311:   begin
[MG3]  312:     mnestr:=Copy(s,1,p-1);
[412]  313:     Delete(s,1,p);
[HK1]  314:     IgnSpc(s);
[2A2]  315:     p:=XPos(',',s);
[R63]  316:     if p=Ø then op1str:=s
[ZRØ]  317:     else
[MUØ]  318:     begin
[EE3]  319:       op1str:=Copy(s,1,p-1);
[412]  32Ø:       Delete(s,1,p);
[3P1]  321:       op2str:=s;
[YK2]  322:       CutSpc(op1str);
[BH2]  323:       IgnSpc(op2str);
[4MØ]  324:     end
[ZNØ]  325:   end;
[9F2]  326:   UpStr(labstr);
[ØG2]  327:   UpStr(mnestr);
[2F2]  328:   UpStr(op1str);
[4F2]  329:   UpStr(op2str);
[WØ3]  33Ø:   Delete(mnestr,1,1);
[ZNØ]  331: end;
[ØØØ]  332:
[WX4]  333: procedure Analyse (s: string);
[CW6]  334: { analysiert eine eingegebene Quelltext-
[9G6]  335: zeile und erzeugt entsprechende Assem-
[DZ2]  336: blersequenzen }
[Z53]  337: var p,l,i: Integer;
[TTØ]  338:     hilf,
[TYØ]  339:     labnr,
[352]  34Ø:     flag: string;
[RYØ]  341:     o1sta,
[UYØ]  342:     o2sta,
[HW2]  343:     doubflg: Boolean;
[ST1]  344:     accu: Char;
[ØØØ]  345:
[MUØ]  346: begin
[Z81]  347:   flag:='';
[EP2]  348:   doubflg:=False;
[WL1]  349:   befcnt:=Ø;
[8T1]  35Ø:   labstr:='';
[CV1]  351:   remstr:='';
[XU1]  352:   mnestr:='';
[RS1]  353:   op1str:='';
[VS1]  354:   op2str:='';
[CM2]  355:   AnalyseLine(s);
[P13]  356:   p:=Pos('_',mnestr);
[8T1]  357:   if p<>Ø then
[MUØ]  358:   begin
[DB3]  359:     l:=Length(mnestr)-p;
[C14]  36Ø:     flag:=Copy(mnestr,p+1,l);
[N63]  361:     Delete(mnestr,p,l+1)
[ZNØ]  362:   end;
[SF2]  363:   ChgReg(op1str);
[VF2]  364:   ChgReg(op2str);
[UT2]  365:   if op2str<>'' then
[MUØ]  366:   begin
[EN3]  367:     o1sta:=TstOp(op1str);
[KN3]  368:     o2sta:=TstOp(op2str);
[CG3]  369:     if NOT (o1sta OR o2sta)
[K43]  37Ø:     then doubflg:=True
[ZNØ]  371:   end;
[DR3]  372:   opnr:=GetMne(mnestr);
[VØ2]  373:   case opnr of
[G8Ø]  374:     1:
[MUØ]  375:     begin
[MW2]  376:       bef[1]:='PUSH AX';
[U24]  377:       bef[2]:='MOV AL,'+op1str;
[E24]  378:       bef[3]:='MOV AH,'+op2str;
[ZR2]  379:       bef[4]:='MUL AH';
[C64]  38Ø:       if NOT TstOp(op1str) then
[MUØ]  381:       begin
[Ø74]  382:         bef[5]:='MOV '+op1str+',AL';
[ST2]  383:         bef[6]:='POP AX';
[RE1]  384:         befcnt:=6
[4MØ]  385:       end
[ZRØ]  386:       else
[MUØ]  387:       begin
[NØ4]  388:         bef[5]:='MOV SS:[Ø],AL';
[ST2]  389:         bef[6]:='POP AX';
[ZE3]  39Ø:         bef[7]:='MOV '+op1str+
[KG1]  391:         ',SS:[Ø]';
[TE1]  392:         befcnt:=7
[4MØ]  393:       end
[ZNØ]  394:     end;
[K8Ø]  395:     2:
[MUØ]  396:     begin
[MW2]  397:       bef[1]:='PUSH AX';
[2X2]  398:       bef[2]:='PUSH DX';
[C44]  399:       bef[3]:='MOV AX,'+op1str;
[T44]  4ØØ:       bef[4]:='MOV DX,'+op2str;
[AU2]  4Ø1:       bef[5]:='MUL DX';
[C64]  4Ø2:       if NOT TstOp(op1str) then
[MUØ]  4Ø3:       begin
[684]  4Ø4:         bef[6]:='MOV '+op1str+',AX';
[3U2]  4Ø5:         bef[7]:='POP DX';
[WT2]  4Ø6:         bef[8]:='POP AX';
[VE1]  4Ø7:         befcnt:=8
[4MØ]  4Ø8:       end
[ZRØ]  4Ø9:       else
[MUØ]  41Ø:       begin
[V14]  411:         bef[6]:='MOV SS:[Ø],AX';
[3U2]  412:         bef[7]:='POP DX';
[WT2]  413:         bef[8]:='POP AX';
[3F3]  414:         bef[9]:='MOV '+op1str+
[KG1]  415:         ',SS:[Ø]';
[XE1]  416:         befcnt:=9
[4MØ]  417:       end
[ZNØ]  418:     end;
[5E2]  419:     3: PutShift(2);
[9E2]  42Ø:     4: PutShift(3);
[DE2]  421:     5: PutShift(4);
[HE2]  422:     6: PutShift(5);
[NE2]  423:     7: PutShift(6);
[TE2]  424:     8: PutShift(7);
[XE2]  425:     9: PutShift(8);
[DF2]  426:     1Ø: PutShift(9);
[XEØ]  427:     11:
[MUØ]  428:     begin
[H33]  429:       Str(repcnt,labnr);
```

```
[2W4]  43Ø:       bef[1]:='RA'+labnr+': '+op1str;
[GV3]  431:       bef[2]:='JNC RB'+labnr;
[ZY3]  432:       bef[3]:='LOOP RA'+labnr;
[V43]  433:       bef[4]:='RB'+labnr+
[242]  434:       ' LABEL NEAR';
[XØ2]  435:       Inc(repcnt);
[LE1]  436:       befcnt:=4
[ZNØ]  437:     end;
[ØFØ]  438:     12:
[MUØ]  439:     begin
[H33]  44Ø:       Str(repcnt,labnr);
[2W4]  441:       bef[1]:='RA'+labnr+': '+op1str;
[NF3]  442:       bef[2]:='JC RB'+labnr;
[ZY3]  443:       bef[3]:='LOOP RA'+labnr;
[V43]  444:       bef[4]:='RB'+labnr+
[242]  445:       ' LABEL NEAR';
[XØ2]  446:       Inc(repcnt);
[LE1]  447:       befcnt:=4
[ZNØ]  448:     end;
[3FØ]  449:     13:
[MUØ]  45Ø:     begin
[MW2]  451:       bef[1]:='PUSH AX';
[HE2]  452:       bef[2]:='LAHF';
[K53]  453:       bef[3]:='AND AH,191';
[AF2]  454:       bef[4]:='SAHF';
[PT2]  455:       bef[5]:='POP AX';
[6M1]  456:       befcnt:=5;
[ZNØ]  457:     end;
[6FØ]  458:     14:
[MUØ]  459:     begin
[MW2]  46Ø:       bef[1]:='PUSH AX';
[HE2]  461:       bef[2]:='LAHF';
[MØ3]  462:       bef[3]:='OR AH,64';
[AF2]  463:       bef[4]:='SAHF';
[PT2]  464:       bef[5]:='POP AX';
[NE1]  465:       befcnt:=5
[ZNØ]  466:     end;
[9FØ]  467:     15:
[MUØ]  468:     begin
[MW2]  469:       bef[1]:='PUSH AX';
[HE2]  47Ø:       bef[2]:='LAHF';
[U33]  471:       bef[3]:='XOR AH,64';
[AF2]  472:       bef[4]:='SAHF';
[PT2]  473:       bef[5]:='POP AX';
[NE1]  474:       befcnt:=5
[ZNØ]  475:     end;
[CFØ]  476:     16:
[MUØ]  477:     begin
[7T2]  478:       if TstOp(op1str)
[L52]  479:       then bef[1]:=
[CN3]  48Ø:       'XOR '+op1str+','+op1str
[GV4]  481:       else bef[1]:='MOV '+op1str+',Ø';
[DE1]  482:       befcnt:=1
[ZNØ]  483:     end;
[FFØ]  484:     17:
[MUØ]  485:     begin
[4B1]  486:       bef[1]:=
[9U4]  487:       'TEST '+op1str+',1 SHL '+op2str;
[DE1]  488:       befcnt:=1
[ZNØ]  489:     end;
[KFØ]  49Ø:     18:
[MUØ]  491:     begin
[ND3]  492:       bef[1]:='OR '+op1str+
[MK2]  493:       ',1 SHL '+op2str;
[DE1]  494:       befcnt:=1
[ZNØ]  495:     end;
[NFØ]  496:     19:
[MUØ]  497:     begin
[TC3]  498:       bef[1]:='AND '+op1str+
[6K3]  499:       ',NOT (1 SHL '+op2str+')';
[DE1]  5ØØ:       befcnt:=1
[ZNØ]  5Ø1:     end;
[WEØ]  5Ø2:     2Ø:
[MUØ]  5Ø3:     begin
[GF3]  5Ø4:       bef[1]:='XOR '+op1str+
[MK2]  5Ø5:       ',1 SHL '+op2str;
[DE1]  5Ø6:       befcnt:=1
[ZNØ]  5Ø7:     end;
[ZEØ]  5Ø8:     21:
[MUØ]  5Ø9:     begin
[MW2]  51Ø:       bef[1]:='PUSH AX';
[YW2]  511:       bef[2]:='PUSH CX';
[4X2]  512:       bef[3]:='PUSH DX';
[YW2]  513:       bef[4]:='PUSH BX';
[WY2]  514:       bef[5]:='PUSH SP';
[UW2]  515:       bef[6]:='PUSH BP';
[TY2]  516:       bef[7]:='PUSH SI';
[ZW2]  517:       bef[8]:='PUSH DI';
[VE1]  518:       befcnt:=8
[ZNØ]  519:     end;
[2FØ]  52Ø:     22:
[MUØ]  521:     begin
[VR2]  522:       bef[1]:='POP DI';
[AT2]  523:       bef[2]:='POP SI';
[NS2]  524:       bef[3]:='POP BP';
[AU2]  525:       bef[4]:='POP SP';
[TT2]  526:       bef[5]:='POP BX';
[1U2]  527:       bef[6]:='POP DX';
[ØU2]  528:       bef[7]:='POP CX';
```

Listing 1. Der Assembler-Präprozessor »asspre.pas« verhilft dem Assembler zu neuen Fähigkeiten

Wer–wo–wann?

Bei Zuschriften bitten wir, folgendes zu beachten: Haben Sie Fragen, Anregungen oder Programme, die Sie der DOS-Redaktion zukommen lassen möchten, adressieren Sie wie folgt:

Redaktion
»DOS International«
Gruberstraße 46c
8011 Poing
Tel. (0 81 21) 7 69-0

Haben Sie Fragen die zum Beispiel Abonnements, Kleinanzeigen oder Databox-Bestellungen betreffen, wenden Sie sich bitte an den Verlag:

DMV-Verlag · DOS International ·
Postfach 250 · 3440 Eschwege
Tel. (0 56 51) 8 09-0
Telefax (0 56 51) 8 09-3 33

```
[WT2]  529:        bef[8]:='POP AX';
[VE1]  53Ø:        befcnt:=8
[ZNØ]  531:      end;
[5FØ]  532:      23:
[MUØ]  533:      begin
[DG2]  534:        for i:=1 to 8 do
[MUØ]  535:        begin
[U72]  536:          Str(i*2,hilf);
[CN4]  537:          bef[i]:='PUSH WORD PTR SS:['
[K71]  538:          +hilf+']'
[ZNØ]  539:        end;
[VE1]  54Ø:        befcnt:=8
[ZNØ]  541:      end;
[8FØ]  542:      24:
[MUØ]  543:      begin
[DG2]  544:        for i:=1 to 8 do
[MUØ]  545:        begin
[8H2]  546:          Str(18-i*2,hilf);
[WM4]  547:          bef[i]:='POP WORD PTR SS:['+
[7A1]  548:          hilf+']'
[ZNØ]  549:        end;
[VE1]  55Ø:        befcnt:=8
[ZNØ]  551:      end;
[BFØ]  552:      25:
[MUØ]  553:      begin
[VØ4]  554:        bef[1]:='MOV SS:[Ø],BP';
[C83]  555:        bef[2]:='MOV BP,SP';
[W33]  556:        bef[3]:='SUB BP,2';
[AB1]  557:        bef[4]:=
[FT4]  558:        'MOV WORD PTR [BP],'+op1str;
[ØØØ]  559:
[393]  56Ø:        bef[5]:='MOV SP,BP';
[RY3]  561:        bef[6]:='MOV BP,SS:[Ø]';
[RE1]  562:        befcnt:=6
[ZNØ]  563:      end;
[ZNØ]  564:    end;
[E82]  565:    if opnr=Ø then
[MUØ]  566:    begin
[L23]  567:      if NOT doubflg then
[MUØ]  568:      begin
[YL1]  569:        befcnt:=1;
[TP2]  57Ø:        bef[1]:=mnestr;
[RT2]  571:        if op1str<>'' then
[E24]  572:          bef[1]:=bef[1]+' '+op1str;
[UT2]  573:        if op2str<>'' then
[SX3]  574:          bef[1]:=bef[1]+','+op2str
[4MØ]  575:      end
[ZRØ]  576:      else
[MUØ]  577:      begin
[1M3]  578:        o1sta:=TstBW(op1str);
[5M3]  579:        o2sta:=TstBW(op2str);
[A45]  58Ø:        if o1sta OR o2sta then accu:='X'
[XD2]  581:        else accu:='L';
[MW2]  582:        bef[1]:='PUSH AX';
[YØ5]  583:        bef[2]:='MOV A'+accu+','+op1str;
[GW3]  584:        bef[3]:=mnestr+' A'+accu+
[CP1]  585:        ','+op2str;
[3V4]  586:        bef[4]:='MOV '+op1str+',A'+accu;
[PT2]  587:        bef[5]:='POP AX';
[NE1]  588:        befcnt:=5
[4MØ]  589:      end
[ZNØ]  59Ø:    end;
[9U2]  591:    if labstr<>'' then
[MUØ]  592:    begin
[HF4]  593:      WriteLn(destin,labstr+':');
[BG1]  594:      Inc(zcnt)
[ZNØ]  595:    end;
[CB2]  596:    if flag<>'' then
[MUØ]  597:    begin
[113]  598:      Str(labcnt,labnr);
[5W5]  599:      WriteLn(destin,'J'+flag+' LA'+labnr);
[A96]  6ØØ:      WriteLn(destin,'JMP SHORT LB'+labnr);
[C94]  6Ø1:      WriteLn(destin,'LA'+labnr+
[S92]  6Ø2:      ' LABEL NEAR');
[C62]  6Ø3:      zcnt:=zcnt+3;
[KW1]  6Ø4:      Inc(labcnt)
[ZNØ]  6Ø5:    end;
[6C3]  6Ø6:    for i:=1 to befcnt do
[144]  6Ø7:    WriteLn(destin,bef[i]);
[M33]  6Ø8:    zcnt:=zcnt+befcnt;
[CB2]  6Ø9:    if flag<>'' then
[MUØ]  61Ø:    begin
[F94]  611:      WriteLn(destin,'LB'+labnr+
[S92]  612:      ' LABEL NEAR');
[BG1]  613:      Inc(zcnt)
[ZNØ]  614:    end;
[SV2]  615:    if remstr<>'' then
[MUØ]  616:    begin
[V34]  617:      WriteLn(destin,remstr);
[BG1]  618:      Inc(zcnt)
[4MØ]  619:    end
[ZNØ]  62Ø: end;
[ØØØ]  621:
[MUØ]  622: begin
[753]  623:   { Hauptprogramm }
[4P1]  624:   repcnt:=Ø;
[ML1]  625:   labcnt:=Ø;
[XF1]  626:   WriteLn;
[FR6]  627:   WriteLn('ASSPRE V1.Ø - Assembler Prepro',
[VL4]  628:   'cessor    (c)1991 DMV Widuch G',
[FY1]  629:   'mbH & Co. KG');
```

```
[XF1]  63Ø:   WriteLn;
[L93]  631:   if ParamCount=Ø then
[MUØ]  632:   begin
[AG6]  633:     WriteLn('Syntax: asspre [d:] [\Pfad',
[UK3]  634:     '] Dateiname [.asp]');
[Y21]  635:     Halt(1)
[ZNØ]  636:   end;
[M63]  637:   file1:=ParamStr(1);
[ZE3]  638:   if Pos('.',file1)=Ø then
[KØ3]  639:     file1:=file1+'.ASP';
[T22]  64Ø:   file2:=file1;
[PL2]  641:   p:=Pos('.',file2);
[S75]  642:   Delete(file2,p,Length(file2)-p+1);
[W22]  643:   file3:=file2;
[CØ3]  644:   file2:=file2+'.ASM';
[Ø13]  645:   file3:=file3+'.NUM';
[F85]  646:   WriteLn('    Quelldatei: ',file1);
[M15]  647:   WriteLn('    Zieldatei : ',file2);
[XF1]  648:   WriteLn;
[WE3]  649:   Assign(source,file1);
[RE3]  65Ø:   Assign(destin,file2);
[FD3]  651:   Assign(number,file3);
[LD2]  652:   Reset(source);
[XT2]  653:   Rewrite(destin);
[VU2]  654:   Rewrite(number);
[P81]  655:   znr:=Ø;
[F91]  656:   zcnt:=Ø;
[R81]  657:   ncnt:=Ø;
[LV3]  658:   while NOT EoF(source) do
[MUØ]  659:   begin
[TS3]  66Ø:     ReadLn(source,linstr);
[NH1]  661:     Inc(znr);
[F54]  662:     if XPos('#',linstr)<>Ø then
[MUØ]  663:     begin
[WT2]  664:       WriteLn(destin);
[X95]  665:       WriteLn(destin,';****Start von:',
[1H2]  666:       ' ',linstr,' ****');
[YS2]  667:       Analyse(linstr);
[825]  668:       WriteLn(destin,';****Ende  von:',
[1H2]  669:       ' ',linstr,' ****');
[P22]  67Ø:       zcnt:=zcnt+3
[4MØ]  671:     end
[ZRØ]  672:     else
[MUØ]  673:     begin
[Ø44]  674:       WriteLn(destin,linstr);
[BG1]  675:       Inc(zcnt)
[ZNØ]  676:     end;
[GT2]  677:     ncnt:=ncnt+zcnt;
[ØA3]  678:     Write(number,zcnt);
[M71]  679:     zcnt:=Ø
[ZNØ]  68Ø:   end;
[AC2]  681:   Close(source);
[7B2]  682:   Close(destin);
[HA2]  683:   Close(number);
[XF1]  684:   WriteLn;
[NH1]  685:   Inc(znr);
[4N1]  686:   Inc(ncnt);
[YY5]  687:   WriteLn(znr,' Zeile(n) bearbeitet, ',
[FA4]  688:   ncnt,' Zeile(n) erzeugt.');
[KNØ]  689: end.
```

Listing 1. Der Assembler-Präprozessor »asspre.pas« verhilft dem Assembler zu neuen Fähigkeiten

```
push   AX
push   DX
mov    AX,CX
mov    DX,2Ø
mul    DX
mov    CX,AX
pop    DX
pop    AX
```

Listing 2. So sieht die Multiplikation von »CX« mit 20 bei normalen Assembler-Befehlen aus

```
[R24]    1: ;Programm: beispiel.asp
[627]    2: ;Funktion: Demonstration des repc-Präfix
[TW5]    3: ;Sprache : AssPre V1.Ø und MASM ab V4.Ø
[8F2]    4: ;          oder TASM
[GW3]    5: ;Autor   : Ingo Kubbilun
[PR4]    6: ;(c)1991 DMV Widuch GmbH & Co. KG
[ØØØ]    7:
[G73]    8: code      segment byte
[ØØØ]    9:
[3X3]   1Ø:          assume  CS:code,DS:code
[ØØØ]   11:
[3M1]   12:          org     Ø1ØØh
[ØØØ]   13:
[693]   14: start:   mov     AX,CS
[1V1]   15:          mov     DS,AX
[3V1]   16:          mov     ES,AX
[DN5]   17:          ;Segmentregister initialisieren
[ULØ]   18:          cld
[EM5]   19:          ;bei Stringbefehl. Zeiger erhöhen
[LN1]   2Ø:          mov     CX,6
```

```
[VE5]    21:             ;Zahlenfolgen beinhalten 6 Zahlen
[FP3]    22:             mov     SI,offset folge1
[SA3]    23:             ;SI zeigt auf Folge 1
[KN3]    24:             mov     DI,offset folge2
[D93]    25:             ;DI zeigt auf Folge 2
[AX1]    26:             #repc   cmpsb
[MF3]    27:             ;Folgen vergleichen
[MF2]    28:             jnc     groesser
[B95]    29:             ;wenn Carry nicht gesetzt, dann
[WF5]    30:             ;ist Folge 1 größer als Folge 2
[3Ø5]    31: kleiner:    mov     DX,offset msg1
[L63]    32:             jmp     short ausgabe
[Y35]    33: groesser:   mov     DX,offset msg2
[ØØØ]    34:
[5Ø3]    35: ausgabe:    push    DX
[3Y1]    36:             mov     AH,Ø9h
[LB3]    37:             mov     DX,offset msg3
[3K1]    38:             int     21h
[XD1]    39:             pop     DX
[3Y1]    4Ø:             mov     AH,Ø9h
[3K1]    41:             int     21h
[3Y1]    42:             mov     AH,Ø9h
[LB3]    43:             mov     DX,offset msg3
[3K1]    44:             int     21h
[B62]    45:             mov     AX,4cØØh
[3K1]    46:             int     21h
[ØØØ]    47:
[ØØØ]    48:
[TØ5]    49: folge1      db      Ø1h,Ø1h,Ø1h,Ø1h,Ø1h,Ø1h
[215]    5Ø: folge2      db      Ø2h,Ø2h,Ø2h,Ø2h,Ø2h,Ø2h
[WH5]    51: msg1        db      "Zahlenfolge 1 ist kle"
[5G4]    52:             db      "iner als Zahlenfolge "
[891]    53:             db      "2.$"
[XM5]    54: msg2        db      "Zahlenfolge 1 ist grö"
[1Y4]    55:             db      "ßer als Zahlenfolge 2"
[E51]    56:             db      ".$"
[H83]    57: msg3        db      Ødh,Øah,"$"
[ØØØ]    58:
[ØØØ]    59:
[112]    6Ø: code         ends
[ØØØ]    61:
[4W1]    62:              end     start
```

Listing 3. »beispiel.asp« demonstriert die Anwendung des Repc-Präfix

```
code        segment byte

            assume  CS:code,DS:code,SS:code

            org     Ø1ØØh

start:      jmp     init

            dw      9Øh dup (?)

init:       cli
            mov     AX,CS
            mov     DS,AX
            add     AX,1Øh
            mov     SS,AX
            mov     SP,Ø1ØØh
            sti
            .
            .
            .



code        ends

            end     start
```

Listing 4. Das Programmgerüst »comstruc.asp« für Com-Dateien

```
code        segment byte

            assume  CS:code,DS:data,SS:stack

start:      mov     AX,data
            mov     DS,AX
            .
            .

code        ends

data        segment byte


data        ends

stack       segment stack

            dw Ø1ØØh dup (?)

stack       ends

            end     start
```

Listing 5. »exestruc.asp« zeigt Ihnen die Organisation einer Exe-Datei für Asspre

```
[NV3]     1: { Programm: edtnum.pas
[CT6]     2:   Funktion: Korrektur der Zeilennummern,
[GU4]     3:             die vom Assembler als fehler-
[7Ø4]     4:             haft ausgegeben werden.
[5Z5]     5:  Sprache  : Turbo Pascal ab Version 4.Ø
[ØR3]     6:   Autor   : Ingo Kubbilun
[485]     7:  (C) 1991 DMV Widuch GmbH & Co. KG }
[ØØØ]     8:
[ØØØ]     9:
[ØA4]    1Ø: program LineNum_Editor;
[TF5]    11: var nbuf: array [1..6ØØØØ] of Byte;
[192]    12:     fname: string;
[PX3]    13:     numhandle: file of Byte;
[WB1]    14:     shandle,
[DB2]    15:     dhandle: Text;
[PL1]    16:     s: string;
[622]    17:     a,i: Integer;
[ØØØ]    18:
[ØØØ]    19:
[A76]    2Ø: procedure AnalyseLine (var s: string);
[KT3]    21: var p,p1,p2,z: Integer;
[R92]    22:     zstr: string;
[ES1]    23:     err: Word;
[MUØ]    24: begin
[M72]    25:   p1:=Pos('(',s);
[U72]    26:   p2:=Pos(')',s);
[684]    27:   zstr:=Copy(s,p1+1,p2-p1-1);
[AR2]    28:   Val(zstr,z,err);
[RR4]    29:   if (err<>Ø) OR (z=1) then Exit;
[9RØ]    3Ø:   p:=1;
[5X1]    31:   while z>Ø do
[MUØ]    32:   begin
[182]    33:     z:=z-nbuf[p];
[BYØ]    34:     Inc(p)
[ZNØ]    35:   end;
[VB2]    36:   Str(p-1,zstr);
[9E3]    37:   Delete(s,p1+1,p2-p1-1);
[573]    38:   Insert(zstr,s,p1+1)
[ZNØ]    39: end;
[ØØØ]    4Ø:
[MUØ]    41: begin
[L93]    42:   if ParamCount=Ø then
[MUØ]    43:   begin
[XF1]    44:     WriteLn;
[TR5]    45:     WriteLn('Edtnum V1.Ø: Datei nicht',
[Ø92]    46:     ' gefunden...');
[T41]    47:     Halt(1);
[ZNØ]    48:   end;
[773]    49:   fname:=ParamStr(1);
[7R5]    5Ø:   if Pos('.',fname)=Ø then fname:=fname+
[Ø31]    51:   '.num';
[G14]    52:   Assign(numhandle,fname);
[MW2]    53:   Reset(numhandle);
[VPØ]    54:   i:=1;
[SG4]    55:   while NOT EoF(numhandle) do
[MUØ]    56:   begin
[824]    57:     Read(numhandle,nbuf[i]);
[XXØ]    58:     Inc(i)
[ZNØ]    59:   end;
[BV2]    6Ø:   Close(numhandle);
[3B4]    61:   Assign(shandle,'ass.inp');
[HL2]    62:   Reset(shandle);
[5C4]    63:   Assign(dhandle,'ass.out');
[WX2]    64:   ReWrite(dhandle);
[TW3]    65:   while NOT EoF(shandle) do
[MUØ]    66:   begin
[CV2]    67:     ReadLn(shandle,s);
[XP4]    68:     if (Pos('(',s)>Ø) AND (Pos(')',s)>Ø)
[AD3]    69:     then AnalyseLine(s);
[933]    7Ø:     WriteLn(dhandle,s)
[ZNØ]    71:   end;
[8K2]    72:   Close(shandle);
[CG2]    73:   Close(dhandle);
[KNØ]    74: end.
```

Listing 6. »edtnum.pas« berechnet korrekte Zeilennummern

```
[FD1]     1: @echo off
[R87]     2: rem Benutzer des Turbo Assemblers ersetzen
[7X6]     3: rem masm durch tasm und link durch tlink !
[HC1]     4: asspre %1
[UØ3]     5: masm %1,,,; >ass.inp
[CC1]     6: edtnum %1
[B62]     7: type ass.out
[U22]     8: erase ass.inp
[142]     9: erase ass.out
[DV1]    1Ø: erase %1.num
[451]    11: link %1;
```

Listing 7. Die Stapeldatei »ass.bat« automatisiert den gesamten Assembliervorgang

Werkstatt Lib/Tlib

Bibliotheken für den Linker

Programme wie Lib und Tlib erlauben den Zusammenschluß zahlreicher einzelner Objektdateien zu großen Bibliotheken. Erfahren Sie im folgenden, welche Vorteile die Verwendung von Bibliotheken mit sich bringt, und wie Sie selbst eigene Bibliotheken aufbauen.

Programme zur Bibliotheksverwaltung wie Lib von Microsoft und Tlib von Borland sind normalerweise Bestandteil großer Compilersysteme, die mit Objektdateien arbeiten.

Der fortgeschrittene und professionelle Programmierer kann damit Unterprogramme oder Funktionen, die in mehreren Programmen Verwendung finden sollen, in Form von Objektdateien auslagern und den Compiler bei seiner Arbeit entlasten. In C beispielsweise können Sie Include-Dateien anstelle von Objektdateien oder Bibliotheken einsetzen. Der Compiler muß den Quellcode dieser Include-Dateien aber bei jedem Compilervorgang erneut übersetzen. Da jedoch Unterprogramme, Funktionen und Prozeduren ihrem Wesen nach statisch sind und praktisch keine Änderungen mehr erfahren, sollten Sie sie als separate Objektdatei kompilieren (Bild 1) und anschließend in eine Bibliothek integrieren (Bild 2). Objektdateien und Bibliotheken spielen dann nur noch für den abschließenden Linklauf eine Rolle. Vor einer detaillierten Auseinandersetzung mit Bibliotheken, wird Ihnen deren »Rohform«, die Objektdatei, vorgeführt. Eine Objektdatei entsteht beim Assemblieren mit leistungsfähigen Assemblern wie MASM oder TASM und beim Übersetzen durch die Compiler zahlreicher Hochsprachen. In einer solchen Objektdatei befinden sich die Unterprogramm- oder Funktionsnamen und der zugehörige Programmcode, allerdings noch nicht in ablauffähigem Zustand. In dieser Form ist beispielsweise schon die Unterscheidung in globale und lokale Variablen getroffen sowie Speichermodell und Segmentnamen definiert. Die Weiterverarbeitung übernimmt in einem separaten Arbeitsgang der Linker, der im Anschluß den »Zwischencode« in die letztlich ablauffähige Version umsetzt (Bild 3).

Bei der Auslagerung von Programmcode in Objektdateien übernimmt der Linker die Aufgabe, die Vollständigkeit und Eindeutigkeit von Variablen-, Unterprogramm- und Funktionsnamen zu überprüfen. Entdeckt der Compiler beim Übersetzen eine mehrmals definierte Funktion, was unter anderem bei der Integration zahlreicher Include-Dateien vorkommen kann, so macht er Sie mit einer Fehlermeldung auf diesen Zustand aufmerksam. Bei extern deklarierten Unterprogrammen hat der Compiler keinen Einfluß auf nicht vorhandene oder mehrfach definierte Symbole. Pragmatisch betrachtet mag es unwesentlich erscheinen, ob nun Assembler und Compiler oder der Linker solche Effekte aufspüren, doch für den Aufbau von Bibliotheken und deren Einsatz sind solche Details sehr wichtig. Wenn Sie ohne integrierte Entwicklungsumgebung arbeiten, wie dies bei fehlender Programmieroberfläche eines Compilers oder kurz vor Abschluß größerer Programmierprojekte mit wenigen Korrekturen der Fall ist, dann reicht es schon aus, daß Sie dem Linker eine erforderliche Objektdatei nicht angegeben haben, um die Codeerzeugung schon vor der Fertigstellung mit einer Fehlermeldung abzubrechen (Bild 4).

Selbst wenn Sie Stapel-, Link- oder Make-Dateien einsetzen, kann es zu solchen Fehlermeldungen kommen, beispielsweise wenn Sie aus Zeit- oder Speicherplatzgründen ein C-Unterprogramm teilweise durch ein Assembler-Unterprogramm mit den zuvor verwendeten Variablennamen ersetzen. Bei einer Vielzahl von einzeln aufgeführten Objektdateien verlieren Sie nicht nur den Überblick, es kann auch vorkommen, daß in der MS-DOS-Eingabezeile der Platz fehlt, um die Aufzählung zu vervollständigen.

Ohne Korrekturen in den Stapel-, Link- oder Make-Dateien erfolgt beim Linklauf dann eine Fehlermeldung wegen nicht aufgefundener Namen (sogenannten Referenzen), da im Beispiel die Assembler-Datei in eine eigene Objektdatei umgesetzt ist, von der die zuvor eigenständige, in C programmierte Objektdatei nun abhängig wird.

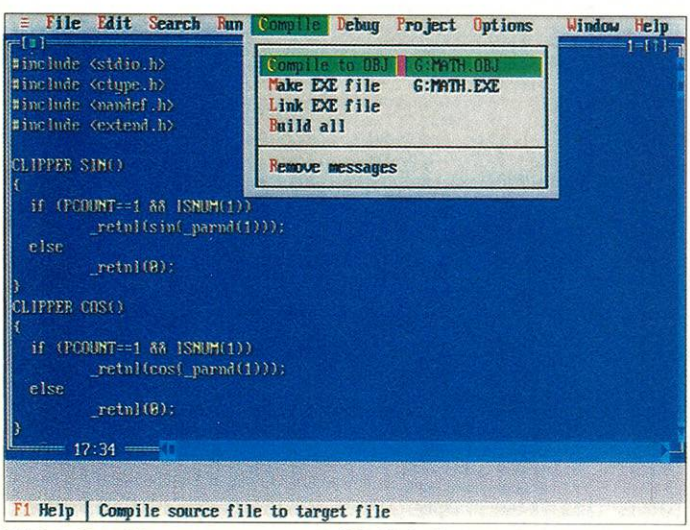


Bild 1. Das Unterprogramm wird lediglich in eine Objektdatei kompiliert (Turbo C + +), es entsteht also keine Com- oder Exe-Datei

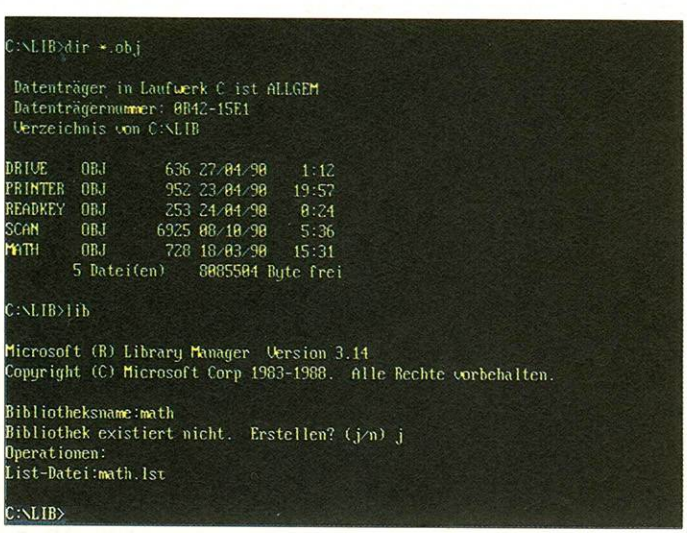


Bild 2. Eine einzelne Objektdatei nehmen Sie mit Lib oder Tlib in eine Bibliothek auf. Ohne Kommandozeilenparameter arbeitet Lib interaktiv.

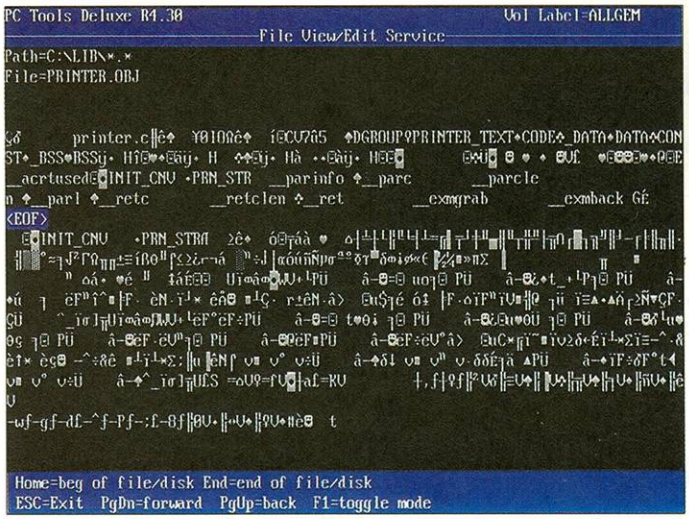


Bild 3. Zu Beginn einer Objektdatei finden Sie unter anderem den Quelldateinamen und einige Referenzen

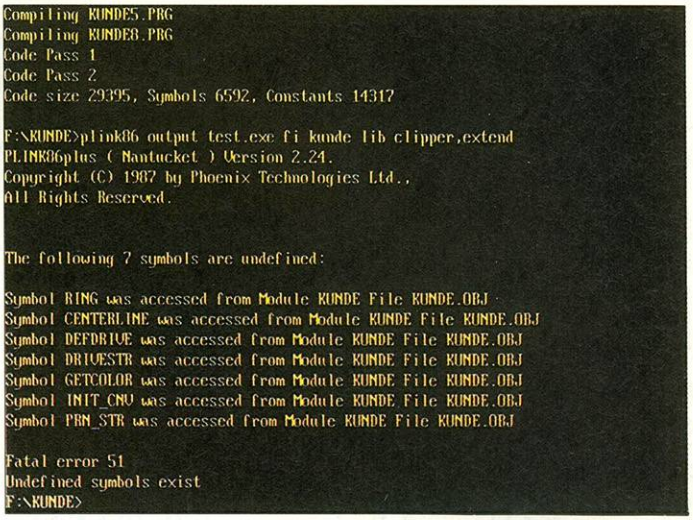


Bild 4. Beim Linkeraufruf wurde unter »fi« (Files) eine Objektdatei nicht aufgeführt

Wären beide Objektdateien in einer Bibliothek integriert, würde der Linker bei der Auswertung aller Referenzen automatisch diesen Zusammenhang »erkennen« und die Informationen beider ehemals eigenständigen Objektdateien ohne Fehlermeldung in ein ausführbares Programm umwandeln.

Die Fähigkeit, verschiedene Programmiersprachen miteinander zu kombinieren, ist ein weiterer Pluspunkt, der für die Verwendung von Objektdateien spricht. Auf diese Weise gehen Sie in einer Sprache nur schwer zu bewältigende Anforderungen in einer dafür besser geeigneten »Umgebung« an. Außerdem können größere Projekte mehrere Programmierer übernehmen, wobei jeder für sich in der Sprache programmiert, die er am besten beherrscht. Lediglich das Hauptprogramm bildet dann den gemeinsamen »roten Faden«.

Objektdateien verschiedener Compiler sind nicht immer kompatibel

Leider erzeugt nicht jede Hochsprache Objektdateien. Turbo Pascal gehört zu dieser Gruppe, hat aber mit dem Konzept der Units eine ähnlich leistungsfähige Lösung anzubieten. Andere Hochsprachen generieren zwar Objektdateien, erfordern aber zwangsläufig, daß das Hauptprogramm in dieser Sprache geschrieben wurde, beispielsweise Clipper oder Turbo Prolog.

Schlimmer jedoch als die fehlende Brücke zwischen einigen Programmiersprachen, die beispielsweise den bequemen Umgang mit Rekursionen und Permutationen von Turbo Prolog oder die Fähigkeiten des problemlosen Zugriffs auf indexsequentielle Daten und deren Verknüpfung mit Relationen oder Filtern von Clipper erlauben würde, ist die Tatsache, daß Objektdateien verschiedener Compilerhersteller im Detail nicht miteinander übereinstimmen. Das ist dadurch bedingt, daß der genaue Aufbau der Objektdateien ein nicht gelüftetes Geheimnis ist und die Programmierer von Compilern, die mit Objektdateien arbeiten, daher versuchen müssen, den Aufbau dieser Dateien selbst zu interpretieren. Aus den entstandenen Detailunterschieden resultiert beispielsweise das Problem, daß Sie die Bibliotheksfunktionen zur Fließkomma-Arithmetik und alle darauf aufbauenden weiteren Funktionen von Turbo C, etwa aus dem Bereich der Grafik, leider nicht unter Clipper verwenden können, da dieser Compiler wiederum unter Microsoft-C entwickelt wurde.

Bibliotheken sind, wie schon erwähnt, eine Zusammenfassung mehrerer Objektdateien. Bei der Integration von Objektdateien in einer Bibliothek verlieren die einzelnen Objektdateien ihre Selbständigkeit als Datei und werden von der Bibliothek lediglich als Name beziehungsweise Referenz verwaltet. Um das zu verdeutlichen, bezeichnet man die in einer Bibliothek integrierten Objektdateien als Module.

Jede im assemblierten oder kompilierten Hauptprogramm verwendete Referenz setzt der Linker beim Binden in die diesem Namen zugehörigen Adressen um. Das folgende Beispiel zeigt, wie dieser Vorgang im Detail abläuft. In der Hochsprache »X«, die mit Objektdateien arbeitet, steht eine Grafikfunktion zur Verfügung, die einen Kreis mit beliebigem Radius an definierter Bildschirmposition bei angegebener Grafikauflösung ausgibt. Beim Einsatz dieser Funktion mit den hierfür benötigten Parametern wandelt der Compiler diese Funktion in eine Reihe von Referenzen um, die alle auf zur Ausführung des Befehls notwendige Unterprogramme verweisen. Im Beispiel könnte eine Unterfunktion des Grafikbefehls zur Übergabe eines Festwertes von π dienen, eine weitere den Kreis selbst berechnen, wieder eine andere die Ausgabe eines Punktes auf dem Bildschirm übernehmen. Doch selbst diese Unterfunktionen können von Fall zu Fall weitere Unterprogramme enthalten. Die vom Compiler erzeugten Referenzen in der Objektdatei des Hauptprogramms beziehen sich in diesem Beispiel lediglich auf einen bestehenden Befehl, also ausschließlich auf die mit dem Compiler gelieferten Standardbibliotheken. Der Linker bindet dann alle zu den Referenzen gehörenden Programmanweisungen hintereinander, also auch den Programmcode einer der zahlreichen Unterprogramme leistungsfähiger Funktionen.

Der Linker bindet häufig überflüssige Teile in die Programme mit ein

Selbst bei einem einzelnen Befehl in einer Hochsprache kann dies in der letztlich erzeugten, ausführbaren Datei etliche KBytes Programmcode ausmachen. Diesen Umstand sollten Sie besonders bei der Kombination mehrerer Programmiersprachen beachten. Programmieren Sie beispielsweise in Clipper und C eine anwenderdefinierte Funktion (»UDF«), die unter anderem Bildschirmausgaben tätigt, so sollten die Ausgaben durch Clipper, also durch das Hauptprogramm, erfolgen. Die Verwendung der Printf-Funktionen von C zur formatierten Ausgabe, wie etwa »printf« oder »cprintf«, vergrößert das erzeugte Programm. Davon abgesehen, daß es bei Clipper nicht unbedingt ratsam ist, Bildschirmmanipulationen außerhalb der Kontrolle des Hauptprogramms durchzuführen, enthält die erwähnte

Familie der C-Funktionen zahlreiche Unterprogramme zur formatierten Ausgabe von Dezimalzahlen, Fließkommawerten, Zeichen etc., wovon viele in diesem Beispiel gar nicht benötigt werden. Eine sich auf »printf« stützende UDF wird demnach weit mehr Code erzeugen, als wenn die Bildschirmausgabe im Beispiel ein Clipper-interner Befehl selbst übernähme. Zur formatierten oder unformatierten Ausgabe von Zeichen und Zahlen stehen unter Clipper »@ x,y say ...« sowie »?« und »??« zur Verfügung, die ohnehin in fast jedem Clipper-Programm einmal auftauchen. Der Linker muß also ohnehin die entsprechenden Teile der Bibliothek in das Programm einbauen. Bei mehrfacher Verwendung von Befehlen und Funktionen integriert der Linker die zugehörigen Teile des Programmcodes nur einmal in das ablauffähige Programm. Lediglich die Parameterübergabe und der Aufruf des entsprechenden Unterprogramms erfolgen jeweils erneut, doch dies beansprucht nur wenig Platz im erzeugten Programm.

Dieses konkrete Beispiel läßt sich für verschiedene Hochsprachen verallgemeinern. Generell ist Vorsicht vor »zu flexiblen« Bibliotheksfunktionen der Compiler geboten, da sich deren unbedachte Nutzung in etlichen KBytes zusätzlichem Programmcode niederschlagen kann. Was bei kleinen Programmen unwesentlich erscheint, spielt bei komplexen Applikationen, die den verbleibenden Hauptspeicher unter anderem auch für interne Tabellen oder Indizes vollständig benötigen, eine große Rolle. Vielleicht zwingt Sie gerade eine leichtfertig verwendete und speicherplatzintensive Funktion der Zusatzbibliothek eines anderen Compilers, ein oder mehrere Overlays zu verwenden, da der verbliebene Hauptspeicher nicht mehr ausreicht. Von zahlreichen, dann üblicherweise erforderlichen Umstellungen in einem Programm, das nicht für den Overlay-Betrieb konzipiert war abgesehen, wird auch die Anwendung durch das Nachladen von Overlays gegenüber dem vorherigen Zustand langsamer. Das Programm müssen Sie schließlich vollständig neu »auf Herz und Nieren« testen.

Aufrufsyntax von Lib und Tlib

lib Bibliotheksname [/pagesize:Anzahl] [Befehl] [,Ausgabedatei] [,neue Bibliothek] [;]

tlib Bibliotheksname [/C] [/E] [/P:] [Befehl] [,Ausgabedatei]

Platzfressende Debug-Informationen sind nur in der Testphase sinnvoll

Zum Kreis der unnötigen Speicherplatzverschwender zählen auch die Debug-Informationen, die in einer komplett geprüften und ausgereiften Objektdatei nicht mehr enthalten sein sollten. Für den Testbetrieb eines Programms ist es sinnvoll, die Zeilennummern für den Debugger bewußt zu aktivieren (Bild 5). Im Fehlerfalle tappen Sie sonst völlig im Dunkeln, in welcher Zeile des Unterprogramms das Problem auftritt, genauer gesagt, ab welcher Stelle es sich bemerkbar macht. Bei fertiggestellten Programmen verschlingen diese Informationen jedoch sinnlos Platz, sowohl in der Objektdatei, wie auch in der Bibliothek und schließlich im Hauptspeicher bei einem damit erzeugten Programm. Die beste Lösung besteht darin, zwei Objektdateien und danach auch zwei Bibliotheken herzustellen, eine mit und eine ohne Debug-Informationen. Zur äußeren Identifikation der unterschiedlichen Bibliotheken ist es ratsam, beispielsweise die Endung »_d« dem Bibliotheksnamen mit Debug-Informationen anzuhängen. Auch die Verwendung verschiedener Speichermodelle, was bei C »zum täglichen Brot« des Programmierers gehört, erfordert das Generieren mehrerer Objektdateien und Bibliotheken. Das Vermischen der Speicherplatzmodelle »tiny«, »small«, »medium«, »large«, »huge« und »compact« führt entweder beim Linklauf (»Fixup overflow in module xxx« bei Tlink) oder auch erst bei der Ausführung des Programms zu komplizierten und auf den ersten Blick selten lösbaren Problemen. Nicht alle Compiler machen Sie auf solche Vorgänge aufmerksam.

So führt beispielsweise der Near-Zugriff auf einen im gleichen Segment erwarteten Namen (Funktion, Prozedur oder dynamische Variable) kombiniert etwa mit »huge« zum Programmabsturz, wenn sich dadurch diese Marke »wider Erwarten« in einem anderen Segment befindet. Achten Sie daher schon bei der Generierung der Bibliothek darauf, daß Sie nur Objektdateien des gleichen Speichermodells darin aufnehmen. Bei Problemen dauert die Suche nach deren Ursache sonst wesentlich länger. Beim Mischen verschiedener Programmiersprachen oder bei der Programmierung unter Benutzeroberflächen müssen Sie ebenfalls beachten, daß die Speichermodelle identisch sind. Bei Unterprogrammen für Clipper ist das Modell »large« zwingend notwendig. Genauere Informationen hierüber sind den jeweiligen Compiler- und Entwicklerunterlagen zu entnehmen.

Schlimmer ist es noch, wenn die mit einem Compiler gelieferten Laufzeitbibliotheken diesbezügliche Fehler enthalten. Als Programmierer vermuten Sie das Problem anfangs in der eigenen »Schöpfung«. Erst die Linkerfehlermeldung, das Durchforsten der Ausgabedatei bei der Generierung, die Überarbeitung oder Einsicht in eine Bibliothek und ein guter Debugger offenbaren dann Situationen, bei denen beispielsweise die Auflösung einer Referenz in einem für den Linker nicht »erwarteten« 64-KByte-Speichersegment liegt und daher nicht oder nicht korrekt erreichbar ist. Sofern der Linker überhaupt Programmcode herstellt, ist ein Programmabsturz beim Erreichen der so umgesetzten Referenz unausweichlich. Glücklicherweise sind solche Compilerfehler die seltene Ausnahme.

Bei der Zuordnung von Objektdateien in Bibliotheken sollten Sie darauf achten, sich aufeinander beziehende Un-

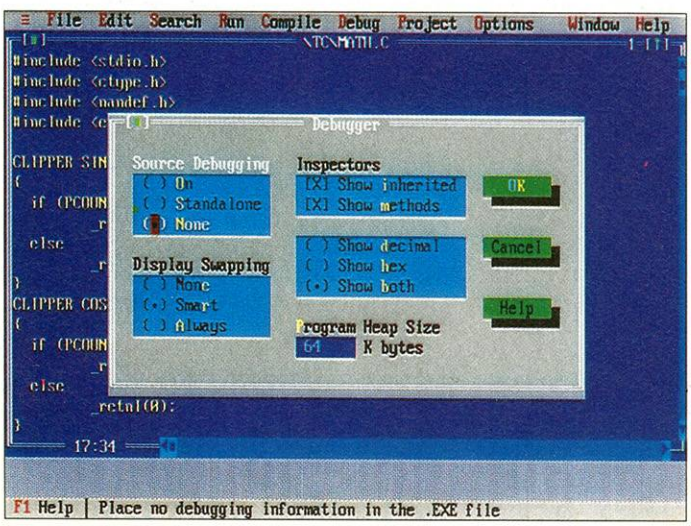


Bild 5. Nach abgeschlossener Testphase einer Objektdatei sollten Sie sie ohne Debug-Informationen kompilieren (am Beispiel von Turbo C + +)

terprogramme in die gleiche Bibliothek einzubinden. Am Beispiel der Grafikfunktion zum Zeichnen eines Kreises in einer beliebigen Hochsprache wurde schon ersichtlich, daß unter Umständen zahlreiche Programme die Routinen bezüglich π oder zum Setzen eines Punktes auf dem Bildschirm nutzen können. Generieren Sie nun mehrere Bibliotheken, in deren Modulen sich Referenzen gegenseitig aufeinander beziehen, so bewirkt dies bei Linkern, die standardmäßig oder überhaupt nur einen Durchgang zum Auflösen aller Referenzen verwenden, unerwartete Schwierigkeiten. Trotz der Präsenz des Moduls in einer der Bibliotheken kann der Linker die Referenz im ersten Durchgang nicht finden. Das Ergebnis des Linklaufs ist somit die Fehlermeldung, daß noch unaufgelöste oder externe Referenzen bestehen. Selbst wenn diese Information ausreicht, um den Linker zu beauftragen, mehrere Suchläufe zur Referenzauflösung zu starten, so bleibt die unschöne Rahmenbedingung, daß der Linker zur Bewältigung seiner Aufgabe wesentlich länger benötigt als erforderlich, insbesondere bei ohnehin langsamen Linkern. Abhilfe schafft das Vertauschen beim Aufruf der Bibliotheken, die manuelle Neuzuordnung von Objektdateien zu Bibliotheken oder aber die Zusammenfassung mehrerer Bibliotheken zu einer einzigen.

Parameter	Bedeutung
Bibliotheksname	Name der Bibliothek, die Sie verwenden oder generieren wollen. Die Angabe der Erweiterung ist nur dann notwendig, wenn es sich nicht um »lib« handelt.
Befehl	Einer oder mehrere der Befehle aus der Übersicht in Tabelle 2
Ausgabedatei	Diese Datei enthält alle in der Bibliothek enthaltenen Unterprogramm-Namen und weist aus, welchem Modul (also einer ehemaligen Objektdatei) sie angehören. Zur Unterstützung bei der Lokalisierung von Fehlern, die bei einer Bibliothek oder dem Zusammenspiel mehrerer auftreten, sollten Sie Ausgabedateien anlegen.

Tabelle 1. Die gemeinsamen Kommandozeilenparameter von Lib und Tlib

Befehl	Bedeutung
+	Aufnahme einer Objektdatei in eine Bibliothek oder Kombination mehrerer Bibliotheken. Standardeinstellung der Dateierweiterung ist »obj«, zum Verschmelzen von Bibliotheken ist die Endung »lib« (oder eine andere gewünschte Endung) erforderlich.
–	Entfernen eines Moduls aus einer Bibliothek, ohne die Objektdatei wieder zu erzeugen
*	Umwandeln eines Moduls in eine Objektdatei gleichen Namens. Das Modul besteht weiterhin in der Bibliothek.
–+	Ersetzen eines Moduls durch eine Objektdatei gleichen Namens
–*	Umwandeln eines Moduls in eine Objektdatei gleichen Namens mit Entfernen des Moduls aus der Bibliothek

Tabelle 2. Alle Lib- und Tlib-Befehle auf einen Blick

Option	Bedeutung
/C	Groß- und Kleinschreibung in Symbolen werden bei der Einbindung einer Objektdatei in eine Bibliothek nicht berücksichtigt. Wenn Sie die Bibliothek mit anderen Linkern als Tlink von Borland verwenden wollen, sollten Sie auf diese Option verzichten.
/E	Tlib generiert einen erweiterten Index der Module einer Bibliothek. Tlink, und nur dieser Linker, kann mit diesen Informationen den Linkvorgang schneller durchführen. Kompatibilität zu anderen Linkern besteht nicht.

Tabelle 3. Die zusätzlichen Optionen, mit denen Sie Tlib steuern können

Fremde Bibliotheken verursachen oft doppelte Referenzen

Sofern Sie mit kommerziell angebotenen Zusatzbibliotheken arbeiten und beim Linklauf Ihrer Programme plötzlich eine Meldung erscheint, daß doppelte Referenzen aufgetreten sind, sollten Sie diese Warnung ernst nehmen. Es ist sehr wahrscheinlich, daß an der Stelle im Programm, an der Sie den entsprechenden Aufruf gestartet haben, ein Programmabsturz erfolgt oder ungewohnte Resultate auftreten. In diesem Fall haben Sie in Ihrem Programm einen Namen vergeben, der auch in der Zusatzbibliothek Verwendung findet. Der Linker hört begreiflicherweise zu suchen auf, wenn er das erste Vorkommen der Referenz gefunden hat. Bringt auch ein Vertauschen der Reihenfolge bei der Auflistung von Objektdateien oder Bibliotheken für den Linker keine Lösung, müssen Sie den doppelten Namen in Ihrem Programm ändern oder auf die Zusatzbibliothek verzichten.

Mit Hilfe all dieser Informationen für die verschiedenen Einsatzgebiete von Bibliotheken gewappnet, sollten Sie sich selbst von deren Effektivität bei Ihren Programmprojekten überzeugen. Zum spontanen Experimentieren ist im Textkasten die Aufrufsyntax der beiden hauptsächlich verwendeten Programme Lib und Tlib dargestellt, Tabelle 1 zeigt die grundsätzlichen Kommandozeilenparameter, und Tabelle 2 widmet sich den Befehlen beider Bibliotheksverwaltungs-Programme.

Der optionale Parameter »/pagesize:« bei Lib beziehungsweise »/P:« bei Tlib steuert die Seitengröße in einer Bibliothek. Jede Bibliothek enthält einen Index, der die Startadresse der einzelnen Module in Seiten enthält. Weil eine Seite standardmäßig 16 Byte hat und ein Indexeintrag 16 Bit lang ist, können Bibliotheken nicht größer als 1 MByte werden. Wenn Sie eine größere Bibliothek brauchen, können Sie die Größe einer Seite bis auf 32768 Byte heraufsetzen. Es muß sich dabei immer um Potenzen von Zwei handeln. In Tabelle 3 sind die zusätzlichen Optionen von Tlib erläutert.

Wenn Sie bei Lib nach dem Namen der Ausgabedatei noch einen Dateinamen angeben, legt der Linker eine neue Bibliothek unter dem angegebenen Namen an. Der abschließende Strichpunkt verhindert, daß Lib Sie bei jedem Aufruf interaktiv nach den Namen der Ausgabedatei und der neuen Bibliothek fragt.

Auch ohne eine kommerzielle Vermarktung oder die Notwendigkeit, Programmier-Know-how zu schützen, werden Sie schnell Freude an der Programmierung und dem Einsatz eigener Bibliotheken finden. Und noch mehr Spaß macht es, ein kompliziertes und lästiges Programmierproblem nicht selbst lösen zu müssen, sondern auf eine vorgefertigte Routine in der Bibliothek eines anderen »Datenverarbeiters« zuzugreifen.

(Herbert Buckel/wn)

Grundlagen Grafikkarten

Ein Bild für die Götter

Vor allem Einsteigern und weniger fortgeschrittenen PC-Besitzern fällt es nicht leicht, sich im Dickicht der Grafikkarten zurechtzufinden. Im folgenden erhalten Sie Einblick in das grundlegende Funktionsprinzip von Grafikkarten und finden Informationen zur Programmierung der einzelnen Bildschirmadapter.

Grafik gehört zu den faszinierenden Anwendungsgebieten der Computertechnik. Bei interessanter Aufbereitung zieht Computergrafik Aufmerksamkeit auf sich. Auf kommerziellem Gebiet eignet sich Grafik hervorragend zur leicht verständlichen Darstellung von Zahlen; im Heimbereich erlaubt sie Computerspiele. Generell führen gute Grafikkarten zu einer benutzerfreundlicheren Bedienungsführung, da sich grafisch wesentlich mehr Informationen übersichtlich und verständlich aufbereiten lassen als auf textorientierten Systemen. Das zeigt sich beispielsweise deutlich an Windows 3.0. Die Ansprüche an Auflösung und Anzahl der darstellbaren Farben haben mit der Zeit zugenommen.

Für den PC gibt es heute fünf Grafikkarten, die sich als Standard etabliert haben: MDA, Hercules, CGA, EGA und VGA. Die steigende Leistungsfähigkeit spiegeln die Daten der Tabelle 1 wider. Der Monochrome Display Adapter »MDA« zeigt nur Text an. Die Hercules-Karte, im Gegensatz zu allen anderen Standards nicht von IBM initiiert, verfügt sowohl über einen Text- als auch über einen Grafikmodus, allerdings nur Schwarzweiß. Der Color Graphics Adapter »CGA« bietet bis zu 16 Farben im Textmodus, bei einer ansprechenden Grafikauflösung allerdings nur vier Farben. Eine EGA-Karte, die Abkürzung für Enhanced Graphics Adapter, erlaubt 16 Farben bei einer gegenüber CGA etwa verdoppelten Grafikauflösung . Der Video Graphics Array »VGA«, die momentan übliche Ausstattung eines PC, bringt bei mittlerer Grafikauflösung immerhin 256 Farben auf den Bildschirm und wird seinem Namen damit einigermaßem gerecht. Viele Hersteller statten ihre VGA-Karten mit höheren maximalen Auflösungen bis zu 1280x1024 Bildpunkten aus, aber da es keinen Standard für die Ansteuerung dieser Modi gibt, verwendet sie ein Großteil der Software nicht. Daneben existiert eine Reihe von speziellen Grafikkarten mit wesentlich höheren Auflösungen und Hunderttausenden gleichzeitig darstellbarer Farben. Aufgrund ihres Preises und der exorbitanten Kosten für einen Monitor, der diese Farbvielfalt und Auflösung verkraftet, sind sie bisher besonderen Anwendungsgebieten vorbehalten.

Die Steuerlogik bringt die Daten aus dem Bildschirmspeicher auf den Monitor

An der Bildschirmausgabe sind mehrere Hard- und Softwarekomponenten beteiligt, die genau aufeinander abgestimmt sein müssen, um eine störungsfreie Darstellung zu garantieren. Für die Bilderzeugung sind drei Bestandteile des PC wichtig: der Monitor, die Steuerlogik und der Bildschirmspeicher. Dazu kommt die Software beziehungsweise das Betriebssystem, dessen Routinen die Bildschirmausgabe maßgeblich beeinflussen. Die Beziehung zwischen den einzelnen Komponenten zeigt Bild 1.

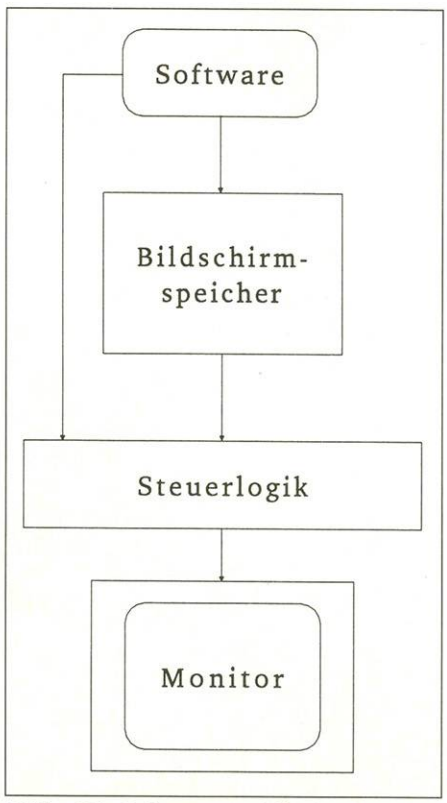


Bild 1. Die Software erreicht den Monitor nur indirekt über Bildschirmspeicher und Steuerlogik

Die Steuerlogik benutzt die Daten im Bildschirmspeicher, um das Signal für den Monitor zu erzeugen. Zuletzt wandelt der Monitor den vom Signalgenerator erzeugten elektrischen Impuls in Licht um.

Normalerweise sind Bildschirmsteuerlogik und Bildschirmspeicher gemeinsam auf dem Bildschirmadapter, auch Grafikkarte genannt, untergebracht. Die Steuerlogik, deren Aufbau Sie in Bild 2 sehen, holt die digitalen Daten aus dem Bildschirmspeicher, übersetzt die Zeichen in Punktmuster und wandelt die digitalen Punkte in ein Signal für den Monitor um.

Der zentrale Baustein, der Videocontroller, bestimmt die Fähigkeiten der Steuerlogik. Er liest die Daten aus dem Bildschirmspeicher und kontrolliert die Funktionen der übrigen Komponenten. Der Zeichengenerator wandelt ein Zeichen vom ASCII-Format in das Punktmuster um, das Sie dann auf dem Bildschirm zu sehen bekommen. Anschließend gibt das sogenannte Schieberegister die Punkte in der korrekten Reihenfolge an den Signalgenerator weiter.

Der Signalgenerator wandelt die Informationen vom Schieberegister in ein Monitorsignal um. Die Verarbeitung der Bildinformationen erfolgt mit einer hohen Geschwindigkeit. Ein Grafikbildschirm mit einer Auflösung von 640x200 Punkten enthält insgesamt 128000 Bildpunkte. Da die Grafikkarte das Bild mindestens 50mal in der Sekunde ausgibt, liefert sie 6,4 Millionen Punkte pro Sekunde. In dieser Rechnung ist die Zeit nicht berücksichtigt, während der der Bildschirm zur Neuausrichtung des Elektronenstrahls kurz abgeschaltet wird. So überrascht es nicht, daß die Übertragungsgeschwindigkeit von der Steuerlogik zum Bildschirm bei Berücksichtigung dieser Ausfallzeiten über 10 Millionen Punkte pro Sekunde beträgt.

Die erste Komponente der Bilderzeugung ist der Bildschirmspeicher, der – wie Sie bereits wissen – die digitalen Daten aufnimmt. Diesen Speicherbereich adressieren Sie genauso wie den normalen Arbeitsspeicher; das bedeutet, daß Sie ohne Schwierigkeiten Daten in ihm ablegen und aus ihm lesen. Der Unterschied zwischen Text- und Grafikdarstellung liegt allein in der Form der Abbildung. Text liegt im Bildschirmspeicher als Folge von ASCII-Zeichen, also von jeweils 1 Byte großen Zahlencodes. Jedem Zeichen folgt zusätzlich ein sogenanntes Attributbyte, das die Farbe oder die Darstel-

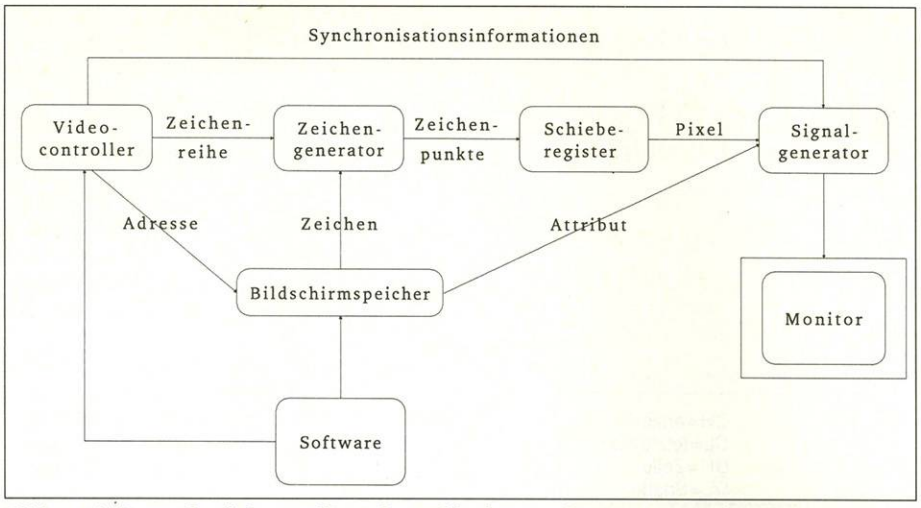


Bild 2. Die Steuerlogik kontrolliert den Bildschirmaufbau und verwaltet den Bildschirmspeicher

lungsform bestimmt. Bei der Grafikausgabe liegen im Bildschirmspeicher die Daten der einzelnen Punkte. Wie bei der Textdarstellung bestimmt die Lage eines Punktes im Bildschirmspeicher dessen Position auf dem Bildschirm. In dem etwas vereinfacht dargestellten Verfahren ist, abhängig von der Anzahl der Farben, ein Punkt in einem, zwei oder mehr Bytes gespeichert.

Neben der Hardware spielt die Software bei der Bildschirmdarstellung eine wichtige Rolle. Es gibt vier Methoden, Daten auf den Monitor zu bringen. Der einfachste Weg, ein Zeichen auf dem Bildschirm abzubilden, ist die Verwendung eines Kommandos in einer höheren Programmiersprache, beispielsweise »PRINT« in Basic, »write« in Pascal oder »printf« in C. Die Ausgabe auf dieser Ebene bietet einige Vorteile. Erstens bleibt sie – sofern die Programmiersprache auch für andere Computer angepaßt wurde – portabel, das Programm funktioniert also auch auf anderen Systemen. Des weiteren ist eine Funktion einer höheren Programmiersprache leicht verständlich. Das erleichtert sowohl die Konzeption als auch die nachfolgende Verbesserung eines Programms erheblich. Die Ausgabe von Daten über eine höhere Programmiersprache weist aber zwei schwerwiegende Nachteile auf. Der wichtigste Nachteil ist der Geschwindigkeitsverlust, der aufgrund der zahlreichen Überprüfungs- und Umrechnungsroutinen auftritt. Der zweite Nachteil besteht in der umständlichen und unvollständigen Unterstützung der Grafikfähigkeiten des PC. Funktionen, die auf Maschinensprachenebene relativ leicht zu realisieren sind, erweisen sich in einer höheren Programmiersprache nur selten als einfach.

Eine andere Art der Zeichenausgabe bietet MS-DOS. Die MS-DOS-Ausgabe-

Nummer	Modus	Grafikauflösung	Textauflösung	Farben	Seiten	Adresse	MDA	Hercules	CGA	EGA	MCGA	VGA	Anmerkung
0	Text	320x200	40x25	16	8	B800	○	○	●	●	●	●	auf CGA monochrom
	Text	320x350	40x25	16	8	B800	○	○	○	●	○	●	
	Text	320x400	40x25	16	8	B800	○	○	○	○	●	○	
	Text	360x400	40x25	16	8	B800	○	○	○	○	○	●	
1	Text	320x200	40x25	16	8	B800	○	○	●	●	●	●	
	Text	320x350	40x25	16	8	B800	○	○	○	●	○	●	
	Text	320x400	40x25	16	8	B800	○	○	○	○	●	○	
	Text	360x400	40x25	16	8	B800	○	○	○	○	○	●	
2	Text	640x200	80x25	16	4	B800	○	○	●	●	○	●	auf CGA monochrom; Modus 2 ist Standard für CGA-, EGA-, MCGA- und VGA-Karte
	Text	640x350	80x25	16	4	B800	○	○	○	●	○	●	
	Text	720x400	80x25	16	4	B800	○	○	○	○	○	●	
3	Text	640x200	80x25	16	4	B800	○	○	●	●	●	●	
	Text	640x350	80x25	16	4	B800	○	○	○	●	○	●	
	Text	640x400	80x25	16	4	B800	○	○	○	○	●	○	
	Text	720x400	80x25	16	4	B800	○	○	○	○	○	●	
4	Grafik	320x200	–	4	1	B800	○	○	●	●	●	●	
5	Grafik	320x200	–	4	1	B800	○	○	●	●	●	●	auf CGA Graustufen
6	Grafik	640x200	–	2	1	B800	○	○	●	●	●	●	
7	Text	720x350	80x25	2	1	B000	●	●	○	●	○	●	
	Text	720x400	80x25	2	1	B000	○	○	○	○	○	●	
8	Grafik	160x200	–	16	2	B800	○	○	○	○	○	○	nur auf IBM-PC-Junior
9	Grafik	320x200	–	16	1	B800	○	○	○	○	○	○	nur auf IBM-PC-Junior
0Ahex	Grafik	640x200	–	4	2	B800	○	○	○	○	○	○	nur auf IBM-PC-Junior
0Bhex	–	–	–	–	–	–	○	○	○	○	○	○	reserviert für das EGA-BIOS
0Chex	–	–	–	–	–	–	○	○	○	○	○	○	reserviert für das EGA-BIOS
0Dhex	Grafik	320x200	–	16	4	A000	○	○	○	●	○	●	
0Ehex	Grafik	640x200	–	16	2	A000	○	○	○	●	○	●	
0Fhex	Grafik	640x350	–	2	1	A000	○	○	○	●	○	●	monochrom
10hex	Grafik	640x350	–	4	1	A000	○	○	○	●	○	○	EGA-Karten mit weniger als 64 KByte RAM
10hex	Grafik	640x350	–	16	1	A000	○	○	○	●	○	●	EGA-Karten mit 64 KByte RAM
11hex	Grafik	640x480	–	2	1	A000	○	○	○	○	●	●	
12hex	Grafik	640x480	–	16	1	A000	○	○	○	○	○	●	
13hex	Grafik	320x200	–	256	1	A000	○	○	○	○	●	●	

Tabelle 1. Die Videomodi zeigen deutlich, wie die fortschreitende Entwicklung der Bildschirmkarten zu höherer Bildschirmauflösung und erweiterter Farbauswahl geführt hat

Register AH	Unter-funktion	Eingabe	Ausgabe	Beschreibung
0		AL=Videomodus (Tabelle 2)	–	legt den Videomodus fest (Text- oder Grafikdarstellung); wenn das 7. Bit von AL gesetzt ist, wird der Bildschirm auf EGA- und VGA-Karten nicht gelöscht
1		CH=000xxxxx (xxxxx=erste Rasterzeile von 0 bis 1Fhex) CL=000xxxxx (xxxxx=letzte Rasterzeile von 0 bis 1Fhex)	–	definiert Cursorgröße
2		BH=Bildschirmseite DH=Zeile (0 bis Zeilenanzahl –1) DL=Spalte (0 bis Spaltenanzahl –1)	–	setzt die Cursorposition; (0,0) entspricht oben links, (Spaltenanzahl –1, Zeilenanzahl –1) entspricht unten rechts
3		BH=Bildschirmseite	CH=erste Rasterzeile CL=letzte Rasterzeile DH=Zeile DL=Spalte	liest Cursorgröße und -position; (0,0) entspricht oben links, (Spaltenanzahl –1, Zeilenanzahl –1) entspricht unten rechts
4			AH=0: keine Lichtgriffel-Position zurückgegeben AH=1: gültige Lichtgriffel-Position zurückgegeben BX=Grafikspalte (0 bis maximale x-Koordinate-1), EGA und VGA CH=Grafikzeile (0 bis maximale y-Koordinate-1), EGA und VGA DH=Zeile des Lichtgriffels (0 bis Zeilenanzahl –1) DL=Spalte des Lichtgriffels (0 bis Spaltenanzahl –1)	liest Position des Lichtgriffels; (0,0) entspricht oben links, (Spaltenanzahl –1, Zeilenanzahl –1) entspricht unten rechts
5		AL=Bildschirmseite	–	wählt aktive Bildschirmseite aus
6		AL=Anzahl der Zeilen (frei werdende Zeilen am Fensterende werden gelöscht) (AL=0: löscht das gesamte Fenster) BH=Farbattribut für die gelöschten Zeilen CH=obere Zeile (0 bis Zeilenanzahl –1) CL=linke Spalte (0 bis Spaltenanzahl –1) DH=untere Zeile (0 bis Zeilenanzahl –1) DL=rechte Spalte (0 bis Spaltenanzahl –1)	–	rollt Bildschirmausschnitt aufwärts
7		AL=Anzahl der Zeilen (frei werdende Zeilen am Fensteranfang werden gelöscht) (AL=0: löscht das gesamte Fenster) BH=Farbattribut für die gelöschten Zeilen CH=obere Zeile (0 bis Zeilenanzahl –1) CL=linke Spalte (0 bis Spaltenanzahl –1) DH=untere Zeile (0 bis Zeilenanzahl –1) DL=rechte Spalte (0 bis Spaltenanzahl –1)	–	rollt Bildschirmausschnitt abwärts
8		BH=Bildschirmseite	Textmodi: AH=Attribut AL=ASCII-Zeichen	liest Zeichen und Attribut (nur Textmodi) an der aktuellen Cursor-Position der angegebenen Bildschirmseite; in den Grafikmodi: AL=0, wenn kein Zeichen erkannt wurde
9		AL=ASCII-Zeichen BH=Bildschirmseite (Videomodus 13hex: BH=Hintergrundfarbe) BL=Attribut (Grafikmodi: BL=Bildpunktfarbe) CX=Anzahl der Zeichen	–	schreibt CX Zeichen mit dem angegebenen Attribut an die Cursorposition der spezifizierten Bildschirmseite (die Cursorposition ändert sich nicht)
0Ahex		AL=ASCII-Zeichen BH=Bildschirmseite (Videomodus 13hex: BH=Hintergrundfarbe) BL=Attribut (Grafikmodi: BL=Bildpunktfarbe) CX=Anzahl der Zeichen	–	schreibt CX Zeichen an die Cursorposition der angegebenen Bildschirmseite (die Cursorposition ändert sich nicht und in den Textmodi bleiben die Attribute erhalten)
0Bhex		Grafikmodus: BH=0: Videomodus 4, 5: BL=Farbe des Bildschirmrahmens und des Hintergrundes	–	wählt Hintergrund-, Rahmen- oder Vordergrundfarbe und Farbpalette aus

Register AH	Unter-funktion	Eingabe	Ausgabe	Beschreibung
		Videomodus 6, 11hex: BL=Vordergrundfarbe BH=1: BL=Farbpalette (Tabelle 3) Textmodus: BH=0: BL=Farbe des Bildschirmrahmens		
0Chex		AL=Farbnummer (je nach Modus 1, 2, 4 oder 8 Bit breit) (Bit 7=1: Xor-Modus) BH=Bildschirmseite (ab EGA-Karte, sonst ignoriert) CX=Grafikspalte (0 bis maximale x-Koordinate) DX=Grafikzeile (0 bis maximale y-Koordinate)	–	setzt Bildpunkt an der bezeichneten Koordinate; wenn Bit 7 von AL gesetzt ist, verknüpft die Routine die Werte mit den Daten im Bildschirmspeicher durch ein exklusives logisches Oder »Xor« (außer im Videomodus 13hex, der 8 Bit benötigt)
0Dhex		BH=Bildschirmseite (ab EGA-Karte, sonst ignoriert) CX=Grafikspalte (0 bis maximale x-Koordinate) DX=Grafikzeile (0 bis maximale y-Koordinate)	AL=Farbnummer	liest Farbe des Bildpunktes an der bezeichneten Koordinate
0Ehex		AL=ASCII-Zeichen BL=Vordergrundfarbe (im Grafikmodus) (bei BIOS-Versionen, die vor 1982 datiert sind, gilt: BH=Bildschirmseite)	–	schreibt Zeichen auf aktive Bildschirmseite und bewegt den Cursor vorwärts (Teletype-Emulation), die ASCII-Zeichen 7, 8, 10 und 13 (Klingel, Rückschritt, Zeilenvorschub und Wagenrücklauf werden ausgeführt, nicht angezeigt)
0Fhex		–	AH=Zeichen pro Zeile AL=Videomodus (Tabelle 2) BH=Bildschirmseite	ermittelt den aktuellen Videomodus
10hex	(Tabelle 4 zeigt, welche Grafikkarten welche Funktionen unterstützen)			setzt Palettenregister sowie Intensität- und Blinkattribut
	AL=0:	BH=Farbwert BL=gewünschtes Palettenregister	–	modifiziert einzelnes Palettenregister
	AL=1:	BH=Farbwert	–	ändert das Overscan-Register
	AL=2:	ES=Segment einer 17-Byte-Tabelle (16 Palettenfarben und 1 Randfarbe) DX=Offset der Tabelle	–	aktualisiert Overscan- und alle Palettenregister
	AL=3:	BL=0 (intensiv) BL=1 (blinkend)	–	legt die Wirkung des Attributbits 7 für den Textmodus fest (intensiv oder blinkend)
	AL=7:	BL=gewünschtes Palettenregister	BH=Farbwert	liest einzelnes Palettenregister
	AL=8	–	BH=Farbwert	liest das Overscan-Register
	AL=9:	ES=Segment einer 17-Byte-Tabelle DX=Offset der Tabelle	Tabelle enthält anschließend die Registerwerte: 10 Palettenfarben und 1 Randfarbe	liest das Overscan- und alle Palettenregister
	AL=16:	BX=Registernummer (0 bis 255) CH=Grünanteil (0 bis 63) CL=Blauanteil (0 bis 63) DH=Rotanteil (0 bis 63)	–	ändert einzelnes Farbregister
	AL=18:	BX=Nummer des ersten Farbregisters CX=Anzahl der zu aktualisierenden Register ES=Segment der Tabelle DX=Offset der Tabelle	–	aktualisiert Farbregisterblock Die Tabelle muß nacheinander die Werte für die verschiedenen Rot-, Grün- und Blauanteile enthalten, jeweils in 3 Byte Rot, Grün und Blau
	AL=19:	BL=0: Seitenmodus wählen BH=0: 4 Seiten mit 64 Farbregistern BH=1: 16 Seiten mit 16 Farbregistern	–	wählt Farbmodus und Bildschirmseite

Register AH	Unter-funktion	Eingabe	Ausgabe	Beschreibung
		BL=1 Seite wählen BH=gewünschte Bildschirmseite		
	AL=21:	BX=Registernummer (0 bis 255)	CH=Grünanteil (0 bis 63) CL=Blauanteil (0 bis 63) DH=Rotanteil (0 bis 63)	liest einzelnes Farbregister
	AL=23:	BX=Nummer des ersten Farbregisters CX=Anzahl der zu lesenden Register ES=Segment der Tabelle DX=Offset der Tabelle	Die Tabelle enthält nacheinander die Werte für die unterschiedlichen Rot-, Grün- und Blauanteile	liest Farbregisterblock
	AL=24:	BL=neuer Wert	–	modifiziert Inhalt des DAC-Mask-Registers, undokumentierte Funktion
	AL=25:	–	BL=gelesener Wert	liest Inhalt des DAC-Mask-Registers, undokumentierte Funktion
	AL=26:	BH=Bildschirmseite	BL=Farbmodus BL=0: 4 Seiten mit 64 Farbregistern BL=1: 16 Seiten mit 16 Farbregistern	liest Farbmodus und Bildschirmseite
	AL=27:	BX=Nummer des ersten Farbregisters CX=Anzahl der umzuwandelnden Farbregister	–	wandelt Farben in Graustufen um
11hex				Zeichengenerator-Routine (nur EGA-, MCGA- und VGA-Karte)
	AL=0:	BH=Anzahl Bytes pro Zeichen (1 bis 32) BL=Speicherblock im Zeichensatzgenerator-RAM (EGA: 0 bis 3, VGA: 0 bis 7) CX=Anzahl neu definierter Zeichen im Zeichensatz DX=ASCII-Code des ersten Zeichens ES=Segment der neuen Zeichensatztabelle BP=Offset der Tabelle	–	lädt benutzerdefinierten Zeichensatz (im Textmodus)
	AL=1:	BL=Speicherblock im Zeichensatzgenerator-RAM	–	lädt 8x14-Punkte-Zeichensatz aus dem ROM
	AL=2:	BL=Speicherblock im Zeichensatzgenerator-RAM	–	lädt 8x8-Punkte-Zeichensatz aus dem ROM
	AL=3:	BL=Auswahl des Speicherblocks Bit 0, 1: Nummer von Block A Bit 2, 3: Nummer von Block B Bit 4: zusätzliches Bit für Block A (nur VGA) Bit 5: zusätzliches Bit für Block B (nur VGA)	–	wählt Zeichensatztabelle aus. Wenn Block A gleich Block B ist, sind es 256 darstellbare Zeichen, wenn Block A ungleich Block B ist, sind es 512.
	AL=4:	BL=Speicherblock im Zeichensatzgenerator-RAM	–	lädt 8x16-Punkte-Zeichensatz aus dem ROM (nur VGA)
	AL=16:	BH=Anzahl Bytes pro Zeichen (1 bis 32) BL=Speicherblock im Zeichensatz-generator-RAM (EGA: 0 bis 3, VGA: 0 bis 7) CX=Anzahl neu definierter Zeichen im Zeichensatz DX=ASCII-Code des ersten Zeichens ES=Segment der neuen Zeichensatztabelle BP=Offset der Tabelle	–	lädt benutzerdefinierten Zeichensatz und programmiert den CRT-Controller
	AL=17:	BL=Speicherblock im Zeichensatzgenerator-RAM	–	lädt 8x14-Punkte-Zeichensatz aus dem ROM und programmiert den CRT-Controller
	AL=18:	BL=Speicherblock im Zeichensatzgenerator-RAM	–	lädt 8x8-Punkte-Zeichensatz aus dem ROM und programmiert den CRT-Controller
	AL=20:	BL=Speicherblock im Zeichensatzgenerator-RAM	–	lädt 8x16-Punkte-Zeichensatz aus dem ROM und programmiert den CRT-Controller, undokumentiert
	AL=32:	ES=Segment der Zeichensatztabelle mit den ASCII-Zeichen 128 bis 255	–	lädt zweite Hälfte des Grafik-Zeichensatzes für die CGA-kompatiblen Grafikmodi

Register AH	Unter-funktion	Eingabe	Ausgabe	Beschreibung
	AL=33:	CX=Anzahl Bytes pro Zeichen BL=Anzahl der Zeilen auf einer Bildschirmseite (Tabelle 5) ES=Segment der neuen Zeichensatztabelle BP=Offset der Tabelle	–	lädt benutzerdefinierten Zeichensatz und macht ihn den grafikorientierten BIOS-Routinen zugänglich
	AL=34:	BL=Anzahl der Zeilen auf einer Bildschirmseite (Tabelle 5)	–	lädt 8x14-Punkte-Zeichensatz aus dem ROM und macht ihn den grafikorientierten BIOS-Routinen zugänglich
	AL=35:	BL=Anzahl der Zeilen auf einer Bildschirmseite (Tabelle 5)	–	lädt 8x8-Punkte-Zeichensatz aus dem ROM und macht ihn den grafikorientierten BIOS-Routinen zugänglich
	AL=36:	BL=Anzahl der Zeilen auf einer Bildschirmseite (Tabelle 5)	–	lädt 8x16-Punkte-Zeichensatz aus dem ROM und macht ihn den grafikorientierten BIOS-Routinen zugänglich
	AL=48:	BH=Zeichensatzspezifikation BH=0: Inhalt des Interrupt-1Fhex-Vektors, Zeiger auf zweite Hälfte des 8x8-Zeichensatzes für CGA-Grafik BH=1: Inhalt des Interrupt-43hex-Vektors, Zeiger auf aktuellen Grafikzeichensatz BH=2: Adresse des ROM-8x14-Punkte-Zeichensatzes BH=3: Adresse des ROM-8x8-Punkte-Zeichensatzes BH=4: Adresse der zweiten Hälfte des ROM-8x8-Zeichensatzes BH=5: Adresse des ROM-9x14-Punkte-Zeichensatzes BH=6: Adresse des ROM-8x16-Punkte-Zeichensatzes (VGA) BH=7: Adresse des ROM-9x16-Punkte-Zeichensatzes (VGA)	CX=Bytes pro Zeichen im aktiven Zeichensatz DL=Anzahl Zeilen pro Seite minus 1 ES:BP=Zeiger auf Zeichensatztabelle (je nach Inhalt von BH vor dem Aufruf)	ermittelt Informationen über einen Zeichensatz
12hex	BL=16:	–	BH=Monitortyp BH=0: Farbgrafikmonitor BH=1: Monochrommonitor BL=Speichergröße der Grafikkarte BL=0: 64 KByte BL=1: 128 KByte BL=2: 192 KByte BL=3: 256 KByte oder mehr CH=Feature-Control-Bits (EGA) CL=Dip-Schalterstellung des Videoadapters	liest Informationen zur Videokonfiguration
	BL=32:	–	–	aktiviert alternative Hardcopy-Routine im Grafikkarten-BIOS
	BL=48:	AL=Anzahl der Rasterzeilen AL=0: 200 Linien (8x8-Zeichen) AL=1: 350 Linien (8x14-Zeichen) AL=2: 400 Linien (9x16-Zeichen)	AL=12hex (wenn Funktion unterstützt wird)	wählt Rasterzeilen für die Textmodi aus
	BL=49:	AL=0: erlaubt das Aktivieren AL=1: läßt das Aktivieren nicht zu	AL=12hex (wenn Funktion unterstützt wird)	kontrolliert das Aktivieren von Standardpaletten
	BL=50:	AL=0: CPU-Zugriff auf Adapter erlauben AL=1: CPU-Zugriff auf Adapter unterbinden	AL=12hex (wenn Funktion unterstützt wird)	regelt den Zugriff auf die Bildschirmkarte
	BL=51:	AL=0: läßt Grauwertumrechnung zu AL=1: deaktiviert die Grauwertumrechnung	AL=12hex (wenn Funktion unterstützt wird)	organisiert das Umrechnen von Farben in Grauwerte

Register AH	Unter-funktion	Eingabe	Ausgabe	Beschreibung
	BL=52:	AL=0: Cursoremulation ein AL=1: Cursoremulation aus	AL=12hex (wenn Funktion unterstützt wird)	dirigiert die Cursoremulation
	BL=53:	AL=0: externe Karte ausschalten AL=1: interne Karte einschalten AL=2: aktive Karte ausschalten AL=3: passive Karte einschalten ES=Segment eines Puffers mit einer Kapazität von 128 Byte, in dem die Statusdaten der Grafikkarte abgelegt werden DX=Offset des Puffers	AL=12hex (wenn Funktion unterstützt wird)	schaltet zwischen zwei eingebauten Grafikkarten um
	BL=54:	AL=0: Bildschirm einschalten AL=1: Bildschirm ausschalten	AL=12hex (wenn Funktion unterstützt wird)	schaltet den Bildschirm ein und aus
13hex		AL=0: Zeichenkette schreiben und Cursorposition nicht verändern AL=1: Zeichenkette schreiben und Cursorposition aktualisieren (Tele type-Emulation, wie bei Funktion 0Ehex) AL=2: Zeichenkette sowie Attribut schreiben und Cursorposition nicht verändern AL=3: Zeichenkette sowie Attribut schreiben und Cursorposition aktualisieren (Teletype-Emulation, wie bei Funktion 0Ehex) BH=Seitennummer BL=Attribut (bei AL=0 oder AL=1) CX=Länge der Zeichenkette DH=Zeile der Schreibposition DL=Spalte der Schreibposition ES=Segment der Zeichenkette BP=Offset der Zeichenkette	–	schreibt eine Zeichenkette auf den Bildschirm
14hex		–	–	reserviert für den IBM-Konvertible
15hex		–	–	reserviert für den IBM-Konvertible
16hex		–	–	reserviert
17hex		–	–	reserviert
18hex		–	–	reserviert
19hex		–	–	reserviert
1Ahex	AL=0:	–	AL=1Ahex (wenn Funktion unterstützt wird) BH=nicht aktiver Monitor (Tabelle 6) BL=aktiver Monitor (Tabelle 6)	ermittelt die Videokonfiguration
	AL=1:	BH=nicht aktiver Monitor (Tabelle 6) BL=aktiver Monitor (Tabelle 6)	AL=1Ahex (wenn Funktion unterstützt wird)	legt die Videokonfiguration fest
1Bhex		BX=0 (Implementationstyp) ES=Segment eines 64-Byte-Puffers DI=Offset des Puffers	AL=1Bhex (wenn Funktion unterstützt wird)	ermittelt den Video- und BIOS-Status (die Tabelle enthält die gewünschten Daten)
1Chex	AL=0:	CX=Status Bit 0=1: Video-Hardware-Status Bit 1=1: Video-BIOS-Datenbereiche Bit 2=1: Video-DAC-Status Bits 3 bis 15: (reserviert)	AL=1Chex (wenn die Funktion unterstützt wird) BX=Puffergröße in 64-Byte-Blöcken	liest die Größe des Videopuffers
	AL=1:	CX=Status (wie in Unterfunktion 0 beschrieben) ES=Segment eines Puffers BX=Offset des Puffers	–	kopiert den Videostatus in den angegebenen Speicherblock
	AL=2:	CX=Status (wie in Unterfunktion 0 beschrieben) ES=Segment des Parameterpuffers BX=Offset des Puffers	AL=1Chex (wenn die Funktion unterstützt wird)	setzt den Videostatus auf die im Puffer gespeicherten Werte

funktionen sind leider sehr einfach und langsam: Sie verarbeiten nur Zeichen. Auf Farben oder irgendwelche anderen Attribute müssen Sie verzichten. Diese Funktionen sind nicht einmal in der Lage, den Cursor an eine bestimmte Stelle zu positionieren. Der Vorteil der Funktionen liegt in ihrer Unabhängigkeit von Hardware und Programmiersprache.
Weitaus mehr Varianten und Funktionen bietet das »BIOS« (Basic Input/Output System) des Rechners. Das BIOS ist in Festwertspeichern, den ROMs in Ihren Computer eingebaut und muß nicht nachgeladen werden. Mit ihm können Sie den Bildschirmmodus festlegen und abfragen, den Cursor positionieren, Farben verwalten, Texte mit dem zugehörigen Attribut ausgeben und einige Hilfsfunktionen ausführen. Zusätzlich gibt es Funktionen zum Setzen und Abfragen eines einzelnen Bildschirmpunktes, da das BIOS grafikfähig ist. Die Funktionen sind schnell und stellen alle notwendigen Fähigkeiten zur Verfügung.
Aber auch die Unterprogramme des BIOS haben zwei Nachteile. Erstens sind die Aufrufe von der Syntax her umständlich und zu schwierig, um sie von einer höheren Programmiersprache aus direkt zu benutzen. Darüber hinaus fehlen Funktionen, die komplexe Aufgaben wie das Zeichnen von Linien oder die Ausgabe eines Kreises erfüllen. Diese Funktionen müssen Sie also selbst programmieren.
Trotz dieser Nachteile ist der Video-Interrupt 10hex des BIOS nicht nur für den Assembler-Programmierer das Tor zum Bildschirm. Er gehört zu den vielfältigsten BIOS-Interrupts. Seine Funktionen erlauben den Zugriff auf alle Bildschirmkarten vom Monochrom- bis zum VGA-Adapter. Sie verarbeiten zum Beispiel Grafik und Text, rollen Bildschirmbereiche auf- oder abwärts und fragen einzelne Bildschirmpositionen auf Textzeichen oder Grafikpunkte ab. Im Zusammenhang mit einer EGA-, MCGA- oder VGA-Karte sind die Routinen für selbstdefinierte Zeichen und die Verwaltung der Farbpaletten erwähnenswert.
Tabelle 1 stellt die unterschiedlichen Videomodi und die zugehörigen Basisinformationen vor. Die Übersicht über mehrere Seiten, Tabelle 2, zeigt alle Funktionen des Video-Interrupts 10hex mit den erforderlichen Ein- und Ausgabeparametern. Tabelle 3 erleichtert den Umgang mit der Funktion 0Bhex, während Tabelle 4 Sie darüber aufklärt, auf welchen Grafikkarten Sie welche der Funktionen 10hex des Video-Interrupts einsetzen können.
Tabelle 5 gibt im Zusammenhang mit Funktion 1Ahex Aufschluß über die Bildschirmkarten- und Gerätekonfiguration. Tabelle 6 beschäftigt sich schließlich mit dem Bereich der Zeichensatzdefinition. Tabelle 7 erklärt den Aufbau des Attributs bei einer monochromen Grafikkarte und Tabelle 8 die Farbverteilung der CGA-Karte.

Direkter Zugriff in Maschinensprache

Die letztgenannte Methode zur Bildschirmansteuerung ist bei weitem die schnellste. Wie eben erwähnt, können Sie direkt in den Bildschirmspeicher schreiben. Dieser Weg ist aber auch der mit Abstand komplizierteste, denn Sie müssen jede Adresse selbst berechnen und das Bitmuster für alle Zeichen und Punkte festlegen. Außerdem kann es Probleme geben, wenn Sie Grafikkarten oder Computer verwenden, die nicht hundertprozentig zum IBM-Standard kompatibel sind. Diese Methode ist also bei jeder Grafikkarte verschieden. Der große Vorteil dieses Verfahrens liegt im Geschwindigkeitsgewinn und der Ausnutzung sämtlicher Fähigkeiten der jeweiligen Grafikkarte. Da sich die Register und Adressen von Adapter zu Adapter geringfügig unterscheiden, sind die eben ausgeführten Erläuterungen nur mit Einschränkung auf jede Bildschirmkarte anwendbar.
Die erste Bildschirmkarte, die für den PC erhältlich war, ist der sogenannte Monochromadapter, abgekürzt MDA. Der Monochromadapter wurde 1981 zusammen mit den ersten IBM-PCs vorgestellt. Die Leistungsfähigkeit der Bildschirmkarte erweist sich als begrenzt, da sie nicht grafikfähig ist. Der MDA kann ausschließlich die standardisierten ASCII-Zeichen und die Sonderzeichen aus dem IBM-Zeichensatz darstellen. Diesen Nachteil gleicht der Monochromadapter allerdings mit seiner hohen Qualität in der Textdarstellung aus, die auch für moderne Grafikkarten Standard geblieben ist. Im Bildschirmspeicher des Monochromadapters sind 2 Byte für jedes Zeichen zuständig. Das erste Byte enthält den ASCII-Wert des jeweiligen Zeichens und das zweite das Attribut. Die Attributangabe, die Tabelle 7 zusammenfaßt, bestimmt, ob der Adapter ein Zeichen blinkend, invers, leuchtend, normal oder unterstrichen darstellt. Der Bildschirmspeicher des Monochromadapters, der an der Speicherstelle B000hex beginnt, ist 4 KByte groß. Das Betriebssystem nutzt aber nur 4000 (80x25x2) Byte des Speicherbereichs aus. Eine Bildschirmzeile von 80 Zeichen Länge belegt also aufgrund des zusätzlichen Attributbyte 160 Byte Speicherplatz. Die Adresse von Zeichen und Attribut können Sie mit den folgenden Formeln ermitteln:

```
Zeichen = B0000hex + 160 x (Zeile -1) +
2 x (Spalte -1)
Attribut = B000hex + 160 x (Zeile -1) +
2 x (Spalte -1)+1
```

Der Monochromadapter hat heutzutage an Bedeutung verloren.
IBM hat sinnvollerweise seine Bildschirmadapter so konstruiert, daß sie immer kompatibel zu ihrem Vorgänger sind. Das heißt, die Programme, die für den Monochromadapter konzipiert wurden, laufen sogar auf einer VGA-Karte ohne Probleme, wenn die Bildschirmausgabe über das BIOS erfolgt. Beachten Sie bitte in der Aufstellung in Tabelle 1 die Sonderstellung der Hercules-Grafikkarte. Der Textmodus der Hercules-Grafikkarte ist voll kompatibel zum Monochromadapter und wird aus diesem Grund auch mit Modus 7 angesprochen. Da die Hercules-Grafikkarte aber keine IBM-Entwicklung ist, unterstützt das BIOS nicht den Grafikmodus.
Zusätzlich zu den Standardversionen gibt es viele Grafikkarten, die weitere Videomodi oder/und Verbesserungen der Originalfassungen bieten. Aufgrund ihrer Bedeutungslosigkeit für den Großteil des PC-Marktes und respektive fehlender Standardisierung seien sie nur kurz erwähnt. Die 1986 dem Publikum vorgestellte »Hercules Graphics Card Plus« zeichnet sich gegenüber der MDA- und der Standard-Hercules-Karte durch verbesserte textorientierte Darstellung aus. Der Programmierer ist in der Lage, selbstdefinierte Zeichensätze zu verwalten, während MDA und Hercules nur auf einen vordefinierten Zeichensatz zugreifen, dessen Aufbau im ROM verankert ist. Eine weitere Verfeinerung der Zeichenausgabe führt dazu, daß textbezogene Anwendungen zwei- bis dreimal schneller ablaufen. Die 1987 auf dem Markt eingeführte »Hercules InColor Card« stellt eine 16-Farben-Version der vorgenannten Grafikkarte

Bit 0 von »BL«	Farbwert	Farbe
0	1	Grün
	2	Rot
	3	Gelb
1	1	Türkis
	2	Magenta
	3	Weiß
Manche CGA-Karten unterstützen zwei weitere Paletten		
2	1	Hellgrün
	2	Hellrot
	3	Gelb
3	1	Helltürkis
	2	Violett
	3	Weiß

Tabelle 3. Die Funktion 0Bhex bestimmt mit Hilfe des BL-Registers eine CGA-kompatible Farbpalette, wenn »BH« den Wert 1 enthält

dar. Der Incolor-Adapter konkurriert mit der EGA-Karte von IBM. Er arbeitet ebenso mit einer Farbpalette aus 64 Farben und steuert den gleichen Monitortyp an.

Die MCGA-Grafikkarte (Abkürzung für »Multi-Color Graphics Array«) findet ausschließlich in den IBM-Modellen 25 und 30 aus der PS/2-Serie Verwendung. Sie emuliert den CGA-Adapter in vielerlei Hinsicht, bietet aber andererseits weitaus facettenreichere Manipulationsvarianten, eine höhere Bildschirmauflösung und mehr Farben. Die MCGA gilt als Vorstufe zur VGA-Karte und läuft wie ihr fortschrittlicher Nachfolger, aufgrund der unterschiedlichen Steuersignale nur auf analogen Monitoren. Die Super-VGA-Karte ist eine Weiterentwicklung der VGA-Karte, die eine höhere Auflösung bietet, allerdings infolge fehlender Standardisierung keine Basis für eine ordentliche Schilderung ihrer Fähigkeiten begründet.

Einigen aktuellen Grafikkarten fehlt die benutzerfreundliche Standardisierung

Als letzter Vertreter seiner Kategorie sei die 8514/A-Karte von IBM erwähnt. Sie erlaubt, in gewisser Hinsicht vergleichbar zu den Super-VGA-Adaptern, eine Auflösung von 1024x768 Bildpunkten und die Auswahl von 256 aus 262144 Farben. Die Programmierung unterscheidet sich grundlegend von den bisher angesprochenen Grafikkarten. Integrierte Chips beinhalten eigene Grafikfunktionen, so daß der Bildschirmadapter Standardaufgaben wie das Ziehen von Linien selbständig übernimmt.

(Markus Zietlow/wn)

Literatur:
Matthias Uphoff: Die Programmierung der EGA/VGA-Grafikkarte, Addison-Wesley 1991, 505 Seiten, ISBN 3-89319-274-3, 79 Mark

Michael Tischer: PC Intern 2.0. Systemprogrammierung, Data Becker Verlag 1990, 1167 Seiten, ISBN 3-89011-331-6, 98 Mark

Thom Hogan: Die PC-Referenz für Programmierer, Systhema Verlag 1988, 535 Seiten, ISBN 3-89390-250-3, 69 Mark

Unterfunktion	EGA	MCGA	VGA	Anmerkung
0	●	●	●	–
1	●	○	●	–
2	●	○	●	–
3	●	●	●	–
4				undokumentiert
5				undokumentiert
6				undokumentiert
7	○	○	●	–
8	○	○	●	–
9	○	○	●	–
0Ahex	○	○	○	nicht standardisiert
0Bhex	○	○	○	nicht standardisiert
0Chex	○	○	○	nicht standardisiert
0Dhex	○	○	○	nicht standardisiert
0Ehex	○	○	○	nicht standardisiert
0Fhex	○	○	○	nicht standardisiert
10hex	○	●	●	–
11hex				undokumentiert
12hex	○	●	●	–
13hex	○	○	●	–
14hex				undokumentiert
15hex	○	●	●	–
16hex				undokumentiert
17hex	○	●	●	–
18hex	○	●	●	–
19hex	○	●	●	–
1Ahex	○	○	●	–
1Bhex	○	●	●	–

Tabelle 4. Die Unterfunktionen der Funktion 10hex des Video-Interrupts 10hex sind nicht auf jedem Grafikadapter zugänglich

Wert	Konfiguration
0	kein Sichtgerät
1	MDA-Karte mit Monochrombildschirm
2	CGA-Karte mit Farbbildschirm
3	(reserviert)
4	EGA-Karte mit Farbbildschirm
5	EGA-Karte mit Monochrombildschirm
6	professioneller Grafikcontroller PGA
7	VGA-Karte mit analogem Monochrombildschirm
8	VGA-Karte mit analogem Farbbildschirm
9	(reserviert)
0Ahex	MCGA-Karte mit digitalem Farbbildschirm
0Bhex	MCGA-Karte mit analogem Monochrombildschirm
0Chex	MCGA-Karte mit analogem Farbbildschirm

Tabelle 6. Die Funktion 1Ahex, die nur auf einer VGA-Karte arbeitet, erleichtert den Umgang mit zwei Sichtgeräten. Diese Kombinationen von Grafikkarten und Monitoren kennt die Routine.

BL	Anzahl der Textzeilen/Bedeutung
0	DL gibt die Anzahl der Textzeilen an
1	14 Textzeilen pro Bildschirm
2	25 Textzeilen pro Bildschirm
3	43 Textzeilen pro Bildschirm

Tabelle 5. Die Register »BL« und »DL« legen fest, wie viele Zeilen pro Bildschirm die BIOS-Routinen verwalten. Das Setzen erfolgt mit der Funktion 11hex, Unterfunktion 21hex des Interrupts 10hex.

Bit-Kombination	dezimaler Wert	Funktion
00000000	0	unsichtbar
00000001	1	unterstrichen
00000111	7	normal
00001001	9	leuchtend und unterstrichen
00001111	15	leuchtend
01110000	112	invers
10000001	129	blinkend und unterstrichen
10000111	135	blinkend
10001001	137	blinkend, leuchtend und unterstrichen
10001111	143	blinkend und leuchtend
11110000	240	blinkend und invers

Tabelle 7. Der Monochromadapter und die Hercules-Karte nutzen Attribute statt Farben zur Hervorhebung von Textbereichen

Nummer	Farbe	Blau	Grün	Rot	Intensiv
0	Schwarz	0	0	0	0
1	Blau	1	0	0	0
2	Grün	0	1	0	0
3	Türkis	1	1	0	0
4	Rot	0	0	1	0
5	Magenta	1	0	1	0
6	Braun	0	1	1	0
7	Hellgrau	1	1	1	0
8	Dunkelgrau	0	0	0	1
9	Hellblau	1	0	0	1
0Ahex	Hellgrün	0	1	0	1
0Bhex	Helltürkis	1	1	0	1
0Chex	Hellrot	0	0	1	1
0Dhex	Hellmagenta	1	0	1	1
0Ehex	Gelb	0	1	1	1
0Fhex	Hellweiß	1	1	1	1

Tabelle 8. Die CGA-Bildschirmkarte erzeugt mit Hilfe der Bit-Planes 16 verschiedene Farben

DOS EXTRA 2
3D-CAD-Programm
DOS-Commander
DOS-Utilities
SForth-Compiler

DOS EXTRA 3
Der PC – glasklar
DFÜ-Programm
Hercules-Grafik
Sprache digitalisiert

DOS EXTRA 4
Speicherresidente Programme
DTP-Programm
EGA-Grafik
MS-DOS intern

DOS EXTRA 5
PC-Grafikkarten
Das BIOS des PC
GW-Basic
Gerätetreiber programmieren

DOS EXTRA 6
Basic im Detail
GW-Basic intern
SBasic-Compiler
Basic-Utilities

DOS EXTRA 7
PC-Komponenten
Sprite-Editor
SBasic Plus
C-Grundlagen

DOS EXTRA 8
Assembler SAM III
Assembler und Hochsprachen
Assembler-Werkstatt
Assembler-Grundlagen

DOS EXTRA 10
MS-DOS 4.01
Hardware-Praxis
Snapshot-Programm
Turbo-Pascal-Units

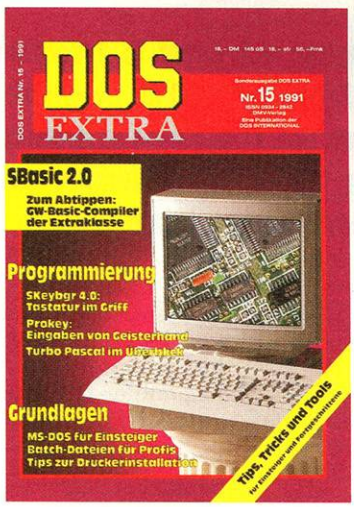
DOS EXTRA 11
Turbo Pascal total
Turbo Pascal intern
SAA-Blockbefehle
MS-DOS-Speicherverwaltung

DOS EXTRA 12
Grafik in Basic
Hardcopy-Utility
Fraktalgrafik in Turbo Pascal
Basic für PC-Grafikkarten

DOS EXTRA 13
Spielekategorien
Die besten Spiele
Abenteuerspiele
Flugsimulator
Sharewarespiele

DOS EXTRA 14
Datensicherungsprogramm mit Benutzeroberfläche
Zeichensatzeditor für 24-Nadel-Drucker
Barcode-Darstellung
aufgeschlüsselt

DOS EXTRA



DOS EXTRA Nr. 15

Aus dem Inhalt der DATABOX:

- der Nachbrenner für GW-Basic: SBasic Version 2.0
- der Tastaturtreiber »Skeybgr« in der Version 4.0
- »Schiffe versenken« mit einem Quick-Basic-Programm
- Simulation von Tastatureingaben mit »Prokey«

Vorsprung für PC-Benutzer mit DOS EXTRA im Abonnement!

4 Ausgaben bei vierteljährlicher Erscheinungsweise mit dem bequemen Zustellerservice per Post:

Inland	DM 60,–
Europa	DM 70,–
Außereuropäisches Ausland	DM 80,–

Keine Bestellprobleme mit dem Dauerbezug der DOS-EXTRA-DATABOX!

Bequemes Dauerbezugsverfahren mit Bankabbuchung und vierteljährlicher Kündigungsmöglichkeit:

Inland	DM 39,–
Europa	DM 41,–
Außereuropäisches Ausland	DM 42,–

Bitte benutzen Sie nur die Bestellkarte!

DATABOX

Die Software zu den DOS-EXTRA-Heften

Sagenhaft günstig, extra viel gute Software fürs Geld! Bringt alle Programme der DOS EXTRA auf Datenträger

Den Aufbau der Databox-Diskette finden Sie in der Datei »read.me« erklärt. Geben Sie deshalb vor dem Start »type read.me« ein. Sämtliche Pascal- und C-Programme liegen auch in einer compilierten Form für Hercules- und Farbgrafikkarten vor. Es ist allerdings nicht gewährleistet, daß diese Versionen auf Hardware, die nicht vollständig kompatibel zum IBM-PC ist, problemlos laufen. In diesem Fall müssen Sie den Quellcode nach der Anpassung mit dem entsprechenden Compiler neu übersetzen.

Die Basic-Programme arbeiten mit dem GW-Basic-Interpreter von Microsoft einwandfrei. Bei Interpretern und Compilern anderer Hersteller kann es zu Problemen kommen, sofern diese nicht vollkompatibel sind.

jeweils **2 Disketten 5 1/4“**
1 Diskette 3 1/2“

35,– DM
Wenn Sie über den DMV-Bestellservice bestellen, gilt folgendes:

Inland:		**Ausland:**	
Einzelpreis	35,– DM	Einzelpreis	35,– DM
zzgl. Versandkosten	4,– DM	zzgl. Versandkosten	6,– DM
Endpreis	**39,– DM**	**Endpreis**	**41,– DM**

Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland 4,– DM bzw. für das Ausland 6,– DM Porto und Verpackung.

Bitte benutzen Sie nur die Bestellkarte!

DMV-Verlag • Postfach 250 • 3440 Eschwege

DMV Daten- und Medienverlag

System-Bibliothek

Assembler wird gesellschaftsfähig

Viele Computeranwender scheuen sich davor, in Assembler zu programmieren. Der Grund liegt vor allem darin, daß im Gegensatz zu Hochsprachen keine fertigen Prozeduren und Funktionen existieren, die zur Programmentwicklung beitragen. Die Routinenbibliothek »system« stellt 40 nützliche Routinen aus dem Programmieralltag bereit.

Wenn Sie zum Beispiel Turbo-Pascal-Kompilate mit Assembler-Programmen gleicher Funktion in puncto Geschwindigkeit vergleichen, stellen Sie fest, daß Assembler-Programme ein viel besseres Laufzeitverhalten aufweisen und darüber hinaus kürzer sind. Dies liegt daran, daß ein Hochsprachen-Compiler ein von Ihnen verfaßtes Programm nicht »eins zu eins« in Assembler umsetzen kann. Hochsprachen-Kompilate sind also nicht so kompakt und schnell wie Assembler-Programme. Der Assembler-Programmierer muß jedoch im Gegenzug auf die leistungsfähigen Prozeduren und Funktionen verzichten, die Hochsprachen zur Verfügung stellen und die auch in seinen Programmen in zyklischen Abständen wiederkehren.
Dieses Problem lösen Sie, indem Sie sich sogenannte Routinenbibliotheken anlegen, die standardisierte Problemlösungen in Form von Assembler-Routinen beinhalten. Diese Bibliotheken (englisch: »libraries«) lassen sich also durchaus mit den bekannten Turbo-Pascal-Units vergleichen.

Die Programmentwicklung mit »system« erleichtert Ihnen die Tagesarbeit

Die Routinenbibliothek »system.asm« (Listing 1) verfügt über 40 Standardroutinen, die Sie in Ihre Assembler-Programme einbinden. Geben Sie zunächst den Quellcode »system.asm« (Listing 1) oder den Objektcode »system.obj« (Listing 2) ein. Im ersten Fall assemblieren Sie das Programm mit der Anweisung

```
masm system,,,;
```

oder

```
tasm system,,,;
```

wenn Sie den Turbo Assembler besitzen. Sie erhalten so ebenfalls die Datei »system.obj«. Danach erzeugen Sie mit dem Library-Manager »lib« beziehungsweise »tlib«, der sich auf Ihrer Assembler-Diskette befindet, die eigentliche Bibliothek.

```
lib system +system;
```

oder:

```
tlib system +system;
```

Dieses Hilfsprogramm unterstützt Sie bei der Erzeugung, Verwaltung und Pflege von Routinenbibliotheken. Es befindet sich nun auf Ihrer Diskette oder Festplatte die Datei »system.lib«, die alle Assembler-Routinen und Variablen aus Listing 1 beinhaltet. In der Tabelle 1 sehen Sie alle verfügbaren Routinen mit Ein- und Ausgabeparametern.
Die beiden Prozeduren »DetectScr« und »InitScr« arbeiten eng zusammen. Die Routine »DetectScr« ermittelt den Typ des angeschlossenen Monitors (Farb- oder Monochrommonitor), legt die Nummer des Bildschirmmodus fest und speichert intern die Segmentadresse des Bildschirmspeichers. Sie gibt im Register AL einen Ergebniswert zwischen Null und Eins zurück, der Ihnen den verwendeten Monitortyp anzeigt. Dieser Rückgabewert dient nur zu Ihrer Information und ist für alle anderen Routinen irrelevant. »InitScr« setzt den ermittelten Bildschirmmodus und initialisiert einige interne Variablen. Sie müssen diese beiden Routinen unbedingt zu Beginn jedes Programms aufrufen, damit die Bildschirmroutinen einwandfrei arbeiten.
Die Bibliothek »system« unterstützt die beiden Bildschirmmodi 3 (Farbe) und 7 (monochrom) und orientiert sich daher an der Auflösung von 80 Spalten und 25 Zeilen. Alle Bildschirmroutinen besitzen darüber hinaus nur Zugriff auf die Bildschirmseite 0. Durch Aufruf der Prozedur »DefWindow« definieren Sie einen »virtuellen« Bildschirm. Ein virtueller Bildschirm besteht aus einem Teil des Gesamtbildschirms. Die Bibliothek verwaltet acht Bildschirme, wobei Sie lediglich die Bildschirme eins bis sieben umdefinieren können. Bildschirm 0 (Standardbildschirm) besitzt einen Sonderstatus und ist nicht modifizierbar. Die Bildschirmverwaltung ordnet des weiteren jedem Bildschirm ein Farbattribut zu, das Sie bereits von den normalen BIOS-Funktionen des Interrupts 10hex kennen und bei der Definition eines Bildschirms im AH-Register angeben. Die unteren vier Bits spezifizieren hierbei die Vordergrundfarbe, die oberen vier Bits die Hintergrundfarbe für Ausgaben auf den Bildschirm. Nach Aufruf der Prozedur »InitScr« setzt diese alle Bildschirme auf 80 Spalten und 25 Zeilen (linke obere Ecke liegt bei 0,0) und versieht jeden Bildschirm mit dem Standardattribut 7 (gebrochenes Weiß auf schwarzem Hintergrund).
Die Routine »SelWindow« selektiert einen Bildschirm. Alle Ausgaben beziehen sich danach nur noch auf den so angewählten virtuellen Bildschirm. Die folgenden Beispiele setzen voraus, daß Sie Bildschirm 1 mit den Werten: Breite 50 Zeichen; Höhe zehn Zeilen; linke obere Ecke bei (10/10) definiert haben. Das Farbattribut ist hierbei nicht von Bedeutung. Beachten Sie bitte, daß alle x-Koordinaten (Spaltenangaben) zwischen 0 und Breite – 1 beziehungsweise alle y-Koordinaten (Zeilenangaben) zwischen 0 und Höhe – 1 innerhalb eines Bildschirms liegen.

Die Bibliothek verfügt über Routinen zur Fensterverwaltung

Die Routine »ClrScr« löscht den aktuellen Bildschirm, das heißt, sie füllt ihn mit dem ASCII-Zeichen auf, das Sie im AL-Register angeben. Sie erreichen einen »normalen« Löschvorgang durch Angabe des ASCII-Zeichens 20hex (Leerzeichen). Die Routine »GotoXY« setzt den Cursor an die angegebene Position auf dem aktuellen Bildschirm. Bei dem oben konstruierten Beispiel sind für die x-Koordinate Werte zwischen 0 und 49, für die y-Koordinate Werte zwischen 0 und 9 erlaubt. Sofern Sie die x-Position größer als 49 angeben, setzt die GotoXY-Routine den Cursor auf Spalte 0 in die nächste Zeile. Wenn Sie die y-Position größer als erlaubt wählen, rollt der Bildschirm so viele Zeilen nach oben, bis die angegebene y-Position auf den größten Zeilenwert normalisiert ist.
Die Prozeduren »PutFrame« und »DelFrame« erlauben Ihnen sogar, Applikationen mit fensterorientierten Be-

Funktionsname	Eingabe	Ausgabe	Beschreibung
DetectScr	–	AL = 0: monochrom AL = 1: Farbe	angeschlossenen Monitortyp ermitteln
InitScr	–	–	Bildschirmmodus und Variablen initialisieren
DefWindow	AH = Nummer (1 bis 7) DH = x-Position DL = y-Position CH = Breite CL = Höhe AL = Standardattribut	Carry-Flag ist gesetzt, wenn ein Fehler auftrat	virtuellen Bildschirm definieren Standardbildschirm nicht modifizierbar
AskWindow	AH = Nummer (0 bis 7)	wie bei »DefWindow«	Koordinaten und Größe eines virtuellen Bildschirms ermitteln
SelWindow	AH = Nummer (0 bis 7)	–	Bildschirm selektieren
ClrScr	AL = Füllzeichen	–	aktuellen Bildschirm löschen
ClrEol	–	–	aktuelle Bildschirmzeile von Cursorposition bis Zeilenende löschen
DelLine	–	–	aktuelle Bildschirmzeile löschen, folgende Zeilen rücken nach
InsLine	–	–	Bildschirmzeile an Cursorposition einfügen
SetColor	AL = Attribut	–	Attribut des aktuellen Bildschirms setzen
GetColor	–	AL = Attribut	Attribut des aktuellen Bildschirms ermitteln
GotoXY	DH = x-Position DL = y-Position	–	Cursor im aktuellen Bildschirm auf Position (x/y) setzen
GetXY	–	wie bei Eingabe »GotoXY«	aktuelle Cursorposition ermitteln
ScrollUp	–	–	aktuellen Bildschirm eine Zeile nach oben rollen
ScrollDn	–	–	aktuellen Bildschirm eine Zeile nach unten rollen
ScrollLf	–	–	aktuellen Bildschirm eine Spalte nach links rollen
ScrollRg	–	–	aktuellen Bildschirm eine Spalte nach rechts rollen
PutFrame	DS:SI = Zeiger auf Frame-Struktur	–	Fenster auf dem Standardbildschirm darstellen
DelFrame	wie bei »PutFrame«	–	Fenster vom Standardbildschirm löschen
Print	DS:SI = Zeiger auf String-Struktur	–	Zeichenkette auf dem aktuellen Bildschirm ausgeben
ReadKey	AH = 0: eingegebenes Zeichen erscheint nicht auf dem Bild AL = 0: Groß- und Kleinbuchstaben AL < > 0: alle Kleinbuchstaben in Großbuchstaben umwandeln DS:SI = Zeiger auf Speicherbereich mit unerlaubten Zeichen CX = Anzahl Zeichen in DS:SI	AH = Tastencode AL = ASCII-Zeichen	Taste vom Benutzer einlesen
Flush	–	–	BIOS-Tastaturpuffer löschen
Input	DS:SI = Zeiger auf String-Struktur AL < > 0: alle Kleinbuchstaben in Großbuchstaben umwandeln AH = 0: String in DS:SI bleibt erhalten AH < > 0: String in DS:SI wird gelöscht, sobald das erste Zeichen eingegeben wird DS:DI = Zeiger auf Speicherbereich mit unerlaubten Zeichen CX = Anzahl Zeichen in DS:DI	AL = 0:Eingabe wurde mit Return abgeschlossen AL = -1: [Esc] wurde gedrückt, String in DS:SI bleibt unverändert DS:SI = Zeiger auf eingegebenen String	Eingabe vom Benutzer abwarten und eingegebene Zeichen in String speichern
DeleteStr	DS:SI = Zeiger auf String-Struktur AH = Position AL = Anzahl Zeichen AL = 0: bis Stringende löschen	DS:SI = Zeiger auf bearbeiteten String	Anzahl Zeichen ab einer bestimmten Position in einem String löschen
InsertStr	DS:SI = Quellstring DS:DI = Zielstring; AH = Position	DS:DI = Zeiger auf bearbeiteten String	Quellstring in Zielstring einfügen
CopyStr	DS:SI = Quellstring DS:DI = Leerstring AH = Position AL = Anzahl Zeichen AL = 0 gesamten String kopieren	DS:DI = Teilstring	Teile eines Quellstrings in einen Leerstring kopieren wenn Zielstring nicht leer: Löschen des Zielstrings
ConcatStr	DS:SI = String 1 DS:DI = String 2	DS:DI = String 2 + String 1	String 1 an String 2 anhängen
PosStr	DS:SI = String DS:DI = Suchstring	AL = Position AL = 0: nicht gefunden	Suchstring in einem anderen String suchen und gegebenenfalls Position zurückgeben
UpStr	DS:SI = String	DS:SI = String	String in Großbuchstaben umwandeln
IgnChar	DS:SI = String AL = ASCII-Code	DS:SI = bearbeiteter String	alle Zeichen (in AL) vom Beginn des Strings überlesen
CutChar	DS:SI = String AL = ASCII-Code	DS:SI = bearbeiteter String	alle Zeichen vom Ende des Strings abschneiden
ConvString	DS:SI = Zeiger auf Zeichenkette DS:DI = Leerstring	DS:DI = String	Zeichenkette in String-Struktur überführen
ParToStr	ES:DI = Zeiger auf Parameter (PSP) (meistens 80hex) DS:SI = Leerstring	DS:SI = String	Parameter (PSP) in String-Struktur überführen
DAdd	DX:AX = Zahl 1 CX:BX = Zahl 2	DX:AX = Ergebnis	zwei 32-Bit-Zahlen addieren
DSub	DX:AX = Zahl 1 CX:BX = Zahl 2	DX:AX = Ergebnis	zwei 32-Bit-Zahlen subtrahieren
DMul	DX:AX = Zahl 1 CX:BX = Zahl 2	DX:AX = Ergebnis	zwei 32-Bit-Zahlen multiplizieren

Funktionsname	Eingabe	Ausgabe	Beschreibung
DDiv	DX:AX = Zahl 1 CX:BX = Zahl 2	DX:AX = Ergebnis CX:BX = Rest Carry-Flag ist gesetzt, bei »Division durch Null«	zwei 32-Bit-Zahlen dividieren Achtung: vorzeichenlose Division
DNeg	DX:AX = Zahl	DX:AX = Ergebnis	Zweierkomplement einer Zahl bilden
NumToStr	DS:SI = Leerstring DX:AX = Zahl BL = Basis (2 bis 16)	DS:SI = Zeiger auf Zahl im String-Format Carry-Flag ist nach Fehler gesetzt	Zahl in String konvertieren
StrToNum	DS:SI = String	DX:AX = Zahl Carry-Flag ist nach Fehler gesetzt	String in Zahl konvertieren hexadezimal »h«, binär »b« und dezimal

Tabelle 1. Alle Bilbliothekroutinen im Überblick

nutzeroberflächen zu entwerfen. Die Routine »PutFrame« stellt ein Fenster auf dem Bildschirm dar, das Sie durch »DelFrame« wieder löschen. Beide Routinen erwarten als Parameter einen Zeiger auf eine Frame-Struktur im Registerpaar »DS:SI«. Tabelle 2 zeigt die Komponenten der Frame-Struktur im Detail. Alle Koordinaten-, Breiten- und Höhenangaben beziehen sich jeweils auf den Standardbildschirm. Der Zeiger bei Adresse 6 und 7 muß auf den Bereich zeigen, in dem Sie die ASCII-Codes des Fensters vor Aufruf von »PutFrame« abgelegt haben. In diesem Bereich legen Sie die komplette Erscheinungsform eines Fensters durch ASCII-Zeichen fest. Die Routine »PutFrame« kopiert die ASCII-Zeichen dieses Bereichs in den Bildschirmspeicher. Das Erscheinungsbild eines Fensters legen Sie durch die Wahl der ASCII-Zeichen fest. Alle Zeichen des Bildschirms, die unter dem dargestellten Fenster verschwinden, speichert »PutFrame« zuvor in einen Puffer, dessen Adresse Sie an Position 8 und 9 der Frame-Struktur angeben. Die Größe des Puffers berechnen Sie mit folgender Formel:

```
Breite*2*Höhe
```

Im Datenbereich des Beispielprogramms »beispiel.asm« (Listing 3) sehen Sie, wie ein einfaches Fenster unter dem Label »win1« definiert wird. Der Vorteil dieser Methode liegt darin, daß Sie auf dem Bildschirm unendlich viele Fenster (nur auf die Größe des Arbeitsspeichers begrenzt) darstellen können, die bereits beim Erscheinen Meldungen oder ähnliches beinhalten. Die Routine »PutFrame« erkennt darüber hinaus drei Spezialcodes im ASCII-Code-Puffer, die Sie an jeder beliebigen Stelle des Puffers angeben können. Der Wert 0 zeigt der Routine an, daß Sie vom normalen Farbattribut der Frame-Struktur abweichen möchten. Die Routine erwartet nach dem Null-Byte ein Farbattribut, auf das sie sich bei der Ausgabe aller folgenden Spalten und Zeilen bezieht. »PutFrame« erwartet deshalb eine 2 Byte lange Angabe. Die Routine betrachtet diese beiden Bytes als Spezialcodes und nicht als ASCII-Zeichen, weshalb sie nicht auf die Breite eines Fensters angerechnet werden. Spezialcode 1 veranlaßt die Routine, an der aktuellen Position kein Zeichen auszugeben – der Hintergrund bleibt also erhalten. Spezialcode 2 erzeugt ebenfalls kein Zeichen auf dem Bildschirm, dunkelt jedoch den Hintergrund an der entsprechenden Stelle ab. So erzeugen Sie Schatteneffekte für Fenster. Betrachten Sie beide Codes (1 und 2) beim Entwerfen eines Fensters wie normale ASCII-Zeichen. Im Beispielprogramm (Listing 3) sehen Sie zur Verwendung der Spezialcodes wiederum eine Frame-Struktur (an Marke »win2«). Die Routine »DelFrame« löscht ein Fenster vom Bildschirm und zeigt den gespeicherten Hintergrund wieder an. Durch Kombination der Prozeduren »DefWindow« und »PutFrame« stellen Sie zum Beispiel ein Fenster auf dem Bildschirm dar und legen in die Umrandung des Fensters einen virtuellen Bildschirm. Indem Sie ihn selektieren, leiten Sie alle Ausgaben in diesen Bildschirm und damit indirekt auch in das Fenster um.

Die Eingaberoutine ist das Herzstück der Systembibliothek

Die Routine »Print« gibt eine Zeichenkette auf den aktuellen Bildschirm aus. Die Zeichenkette muß hierbei als String-Struktur vorliegen. Die Struktur »string« sehen Sie in Tabelle 3. Das erste Byte der Struktur beinhaltet die Zeichen, die der String maximal aufnehmen kann (1 bis 255). Das zweite Byte gibt die aktuelle Länge des Strings in Zeichen wieder. Mit dem dritten Byte der Struktur »string« beginnt der Inhalt der Zeichenkette. Mögliche Stringdefinitionen sehen so aus:

```
str1 db 80,0,80 dup (?)
str2 db 255,3,"DOS",252 dup (?)
```

Im ersten Fall definieren Sie einen Leerstring mit der Bezeichnung »str1«, der 80 Zeichen aufnehmen kann. »str2« dagegen darf maximal 255 Zeichen beinhalten. Seine aktuelle Länge beträgt drei Buchstaben, also das Wort »DOS«. Die Prozedur »ReadKey« wartet auf einen Tastendruck des Benutzers und gibt den Tastencode mit zugehörigem ASCII-Code im AX-Register zurück. Das Ergebnis ist der gleiche Wert, den Sie nach Aufruf des Tastatur-Interrupts 16hex, Funktion 0 erhalten. Wie aus Tabelle 1 hervorgeht, rufen Sie »ReadKey« mit vielen Parametern auf, die diese Routine so komfortabel machen. Die Prozedur »Flush« löscht den BIOS-Tastaturpuffer, so daß Sie zum Beispiel vor Sicherheitsabfragen bereits gedrückte Tasten aus dem Tastaturpuffer eliminieren können.

Byte	Bedeutung
0	x-Position (relativ zum Standardbild)
1	y-Position (relativ zum Standardbild)
2	Höhe (in Zeilen)
3	Breite (in Spalten)
4	Farbattribut
5	auf 1 setzen
6 und 7	Zeiger auf Puffer mit ASCII-Zeichen
8 und 9	Zeiger auf Puffer für Hintergrund

Tabelle 2. Die Datenstruktur »frame« im Detail

Mit der Prozedur »Input« steht Ihnen eine Eingabefunktion zur Verfügung, die nahezu keine Wünsche mehr offen läßt. Im Registerpaar »DS:SI« übergeben Sie den Zeiger auf eine String-Struktur. Ist der String beim Aufruf der Routine nicht leer, so gibt »Input« den Inhalt auf den Bildschirm aus. Durch das AL-Register bestimmen Sie, ob die Routine alle eingegebenen Zeichen automatisch in Großbuchstaben umwandelt (»AL« ist ungleich 0) oder Groß- und Kleinschreibung toleriert (»AL« gleich 0). Der Wert im AH-Register besitzt eine besondere Funktion und zeigt nur Auswirkungen, wenn Sie im Registerpaar »DS:SI« einen Zeiger auf einen String übergeben, der bereits vor der Eingabe Zei-

Byte	Bedeutung
0	maximale Anzahl Zeichen
1	aktuelle Anzahl Zeichen
2 bis 255	Inhalt des Strings

Tabelle 3. Zeichenketten müssen Sie im Format »string« übergeben

chen beinhaltet. Wenn Sie den Wert 0 übergeben, bleibt dieser String auf jeden Fall im Eingabepuffer erhalten und steht zum Editieren zur Verfügung. Bei Angabe eines Wertes ungleich 0 löscht die Inputroutine diesen String, sobald Sie das erste ASCII-Zeichen eingeben. Sollten Sie diesen vorgegebenen String jedoch vor der Eingabe eines Zeichens bearbeitet haben, bleibt der String ebenfalls erhalten.

Der Eingabe können Sie einen Text zur Bearbeitung vorgeben

Sie kennen diese Einrichtung von vielen Programmen, die zum Beispiel die Eingabe eines Dateinamens erwarten. Viele Programme geben Ihnen eine Dateimaske oder einen Dateinamen vor, den Sie durch Drücken der Eingabe-Taste [Enter] übernehmen. Sobald Sie sich für einen anderen Dateinamen entscheiden und den ersten Buchstaben des Namens eintippen, verschwindet diese Vorgabe. Die Routine bietet Ihnen darüber hinaus die Funktion, unerlaubte Zeichen von der Eingabe auszuschließen. Legen Sie hierzu alle unerlaubten Zeichen in einem Puffer ab und übergeben Sie die Adresse des Puffers im Registerpaar »DS:DI«. Das CX-Register müssen Sie zusätzlich mit der Anzahl Zeichen in diesen Puffer laden. Alternativ hierzu geben Sie den Wert 0 im CX-Register an, wenn die Eingabe alle Zeichen umfassen darf (betrachten Sie hierzu auch die Prozedur »ReadKey«).

Die Routine erlaubt das Benutzen aller vier Pfeiltasten, das Löschen durch die beiden Löschtasten sowie das Löschen des gesamten Strings durch die Tastenkombination [Strg-Pos1]. Mit der Taste [Einfg] wechseln Sie zwischen dem Einfüge- und Überschreibmodus. Die Taste [Pos1] positioniert den Cursor an den Stringanfang, [Ende] an das Ende des Strings. Durch Betätigen der Eingabe-Taste [Enter] schließen Sie Ihre Eingabe ab. In diesem Fall gibt Ihnen die Routine den Wert 0 im AL-Register zurück. Wenn Sie die Abbruch-Taste [Esc] betätigen, erhalten Sie im AL-Register den Wert -1 zurück, und der eventuell vorgegebene String, auf den »DS:SI« zeigt, ist unverändert.

Die Bibliothek verfügt über acht Stringmanipulationsbefehle, die es in ähnlicher Form bei der Programmiersprache Turbo Pascal gibt. Allen Routinen ist gemein, daß Sie ihnen einen oder zwei Zeiger auf String-Strukturen übergeben müssen. Die detaillierten Übergabeparameter finden Sie in Tabelle 1. Wenn Sie sich auf Positionen innerhalb eines Strings beziehen, dürfen diese zwischen 1 und der Länge des Strings liegen. Die Prozedur »ConvString« überführt eine Zeichenkette, die mit einer 0 abgeschlossen ist, in das Stringformat. Diese Routine erleichtert Ihnen die Arbeit mit der String-Struktur, da Sie zum Beispiel die Länge einer Zeichenkette nicht selbst ausrechnen müssen. Sie definieren hierzu einfach eine Zeichenkette und einen Leerstring:

```
str1 db "DOS International",0
str2 db 30,0,30 dup (?)
```

und überführen die Zeichenkette in das Stringformat:

```
mov si,offset str1
mov di,offset str2
call ConvString
```

Die Routine »ParToStr« arbeitet ähnlich wie »ConvString«. Sie konvertiert jedoch die Parameter, die MS-DOS einem aufgerufenen Programm an Adresse 80hex im PSP (Program Segment Prefix) übergibt, in das Stringformat.

Berechnungen nehmen Sie mit 32 Bit starken Komponenten vor

Die Bibliothek »system« unterstützt Sie auch bei Rechenoperationen im 32-Bit-Bereich und stellt hierzu die Routinen »DAdd«, »DSub«, »DMul«, »DDiv« und »DNeg« (»D« steht für »double«) zur Verfügung. Die Prozeduren »NumToStr« und »StrToNum« unterstützen Sie bei der Konvertierung von Zahlen in Strings und umgekehrt. Die Routine »NumToStr« erwartet im BL-Register die Basis der Zahl (2 bis 16), mit deren Hilfe Sie eine Zahl in allen Zahlensystemen darstellen können - zum Beispiel 2 (binär), 10 (dezimal), 16 (hexadezimal) etc. Die Routine »StrToNum« stellt das Gegenstück zu »NumToStr« dar. Sie konvertiert eine Zahl, die als String vorliegt, in den äquivalenten Zahlenwert. Hierbei unterstützt sie die Standardzahlensysteme Binär, Dezimal und Hexadezimal. Einer binären Zahl müssen Sie ein »b« und einer hexadezimalen Zahl ein »h« nachstellen, damit die Routine zwischen den Zahlensystemen differenzieren kann. Bei dezimalen Zahlen erlaubt die Routine darüber hinaus das Voranstellen eines Minus-Zeichens.

Das Beispielprogramm (Listing 3) verdeutlicht das Benutzen der Bibliotheksroutinen in eigenen Assembler-Programmen. Listing 4 (»leer.asm«) zeigt Ihnen das Programmgerüst für die Benutzung der Bibliothek »system«. Sie müssen unbedingt zwei Segmente mit den Bezeichnungen »code« und »data« deklarieren, da der Linker sonst keine der Routinen in den Objektcode implementiert. Die beiden Listings 5 und 6 stellen Include-Dateien dar, die alle Extrn-Deklarationen für die Benutzung der Bibliothek beinhalten. Die Datei »sysdata.inc« müssen Sie unbedingt in das Datensegment einfügen, während die Datei »syscode.inc« (im Codesegment unterbringen) für das Linken der Bibliotheksroutinen verantwortlich ist. »syscode.inc« sorgt dafür, daß der Linker alle Routinen der Bibliothek einbindet. Sie können jedoch alternativ hierzu nur die benötigten Routinen im Quelltext als »extrn« deklarieren. Jedes Assembler-Programm, das Routinen aus der Bibliothek »system« benötigt, müssen Sie folgendermaßen assemblieren:

```
masm <Programmname>,,,;
link <Programmname>,,,system;
```

beziehungsweise

```
tasm <Programmname>,,,;
tlink <Programmname>,,,system;
```

Der Linker bindet hierdurch nur die verwendeten Routinen in den Objektcode des Programms ein. Der MASM wird Sie zumindest hinsichtlich der Benutzung der Marke »Width« warnen. Der MASM bemängelt außerdem bei der Assemblierung von »beispiel.asm« die Belegung von »Loop«. Beim TASM kommen die Marken »Stack« und »Instr« hinzu.

(Ingo Kubbilun/db)

```
[2K3]    1: ;Programm: system.asm
[637]    2: ;Funktion: Routinen-Bibliothek zum Einbin-
[W75]    3: ;           den in Assembler-Programme
[PM4]    4: ;Sprache : MASM ab Version 4.Ø
[TP1]    5: ;           TASM
[GW3]    6: ;Autor    : Ingo Kubbilun
[7X4]    7: ;(c) 1991 DMV Widuch GmbH & Co. KG
[ØØØ]    8:
[4T3]    9: ; Public-Deklarationen
[ØØØ]   1Ø:
[LY1]   11: ; Variablen
[5W6]   12: public ScrMode, ScrSeg, WinYPos, WinXPos
[HK6]   13: public Height, Width, ActY, ActX, WinAttr
[TW6]   14: public CurFlag, WSelect, Windows, KeyFlag
[XT5]   15: public BufAdr, Insert, StrBuf, BufPos
[2C6]   16: public AnzKeys, PosXY, SysStr, OutFlag
[6K6]   17: public DelFlag, Basis, NegFlag, ScrUpFlag
[ØØØ]   18:
[E32]   19: ; Prozeduren
[HN6]   2Ø: public DetectScr, InitScr, ClrScr, ClrEol
[VH5]   21: public DelLine, InsLine, SetColor
[MA6]   22: public GetColor, DefWindow, AskWindow
[536]   23: public SelWindow, PutFrame, DelFrame
[Ø26]   24: public ScrollUp, ScrollDn, ScrollLf
[EC6]   25: public ScrollRg, GotoXY, GetXY, Print
[DL6]   26: public ConvString, ReadKey, Flush, Input
[ØA6]   27: public DeleteStr, InsertStr, CopyStr
```

Listing 1. »system.asm« stellt Ihnen 40 nützliche Routinen zur Verfügung

```
[WU6]   28: public ConcatStr, PosStr, UpStr, IgnChar
[FY5]   29: public CutChar, ParToStr, DAdd, DSub
[RD5]   3Ø: public DMul, DDiv, DNeg, NumToStr
[DT2]   31: public StrToNum
[ØØØ]   32:
[8F4]   33: Data        segment word public
[ØØØ]   34:
[AB2]   35: ScrMode     db ØØh
[R32]   36: ScrSeg      dw ?
[RA2]   37: WinYPos     db ?
[PA2]   38: WinXPos     db ?
[U12]   39: Height      db ?
[1Y1]   4Ø: Width       db ?
[SV1]   41: ActY        db ?
[RV1]   42: ActX        db ?
[KA2]   43: WinAttr     db ?
[E72]   44: CurFlag     db ?
[E82]   45: WSelect     db ?
[2F3]   46: Windows     db 4Øh dup (?)
[A72]   47: KeyFlag     db ?
[832]   48: BufAdr      dw ?
[H42]   49: Insert      db ?
[473]   5Ø: StrBuf      db 26Ø dup (?)
[632]   51: BufPos      db ?
[EC2]   52: AnzKeys     dw ?
[H22]   53: PosXY       dw ?
[8N2]   54: SysStr      db 1,1,?
[E82]   55: OutFlag     db ?
[852]   56: DelFlag     db ?
[6Y1]   57: Basis       db ?
[852]   58: NegFlag     db ?
[1C2]   59: ScrUpFlag   db ?
[ØØØ]   6Ø:
[N12]   61: Data        ends
[ØØØ]   62:
[XE4]   63: Code        segment byte public
[7Z3]   64:             assume  CS:Code, DS:Data
[ØØØ]   65:
[HM3]   66: DetectScr   proc    near
[1G1]   67:             push    CX
[PF1]   68:             push    ES
[PV1]   69:             xor     CX,CX
[5V1]   7Ø:             mov     ES,CX
[A43]   71:             mov     CX,ES:[Ø463h]
[562]   72:             cmp     CX,Ø3D4h
[GD1]   73:             pop     ES
[VD1]   74:             pop     CX
[SS1]   75:             jz      Color
[LC5]   76: Mono:       mov     byte ptr ScrMode,7
[WØ3]   77:             mov     ScrSeg,ØB ØØØh
[MT1]   78:             xor     AL,AL
[3PØ]   79:             ret
[DF5]   8Ø: Color:      mov     byte ptr ScrMode,3
[M13]   81:             mov     ScrSeg,ØB8ØØh
[6M1]   82:             mov     AL,1
[3PØ]   83:             ret
[EL2]   84: DetectScr   endp
[ØØØ]   85:
[DC3]   86: InitScr     proc    near
[ZF4]   87:             mov     AL,byte ptr ScrMode
[3L1]   88:             or      AL,AL
[152]   89:             jnz     ModeOk
[3PØ]   9Ø:             ret
[G63]   91: ModeOk:     xor     AH,AH
[YH1]   92:             int     1Øh
[WZ3]   93:             mov     SI,offset Windows
[Z94]   94:             mov     DI,offset Windows+8
[WS3]   95:             mov     word ptr [SI],Ø
[HH4]   96:             mov     word ptr [SI+2],5Ø19h
[GØ4]   97:             mov     word ptr [SI+4],Ø
[414]   98:             mov     word ptr [SI+6],7
[ZXØ]   99:             pushf
[PF1]  1ØØ:             push    ES
[ULØ]  1Ø1:             cld
[MF1]  1Ø2:             push    DS
[GD1]  1Ø3:             pop     ES
[CØ2]  1Ø4:             mov     CX,1Ch
[LZ1]  1Ø5:             rep     movsw
[GD1]  1Ø6:             pop     ES
[LSØ]  1Ø7:             popf
[NL2]  1Ø8:             mov     WSelect,Ø
[8P2]  1Ø9:             jmp     SelectWin
[SE2]  11Ø: InitScr     endp
[ØØØ]  111:
[HA3]  112: ClrScr      proc    near
[ZXØ]  113:             pushf
[PF1]  114:             push    ES
[HM4]  115:             mov     DX,word ptr WinYPos
[VV2]  116:             call    BerechneXY
[KS2]  117:             mov     AH,WinAttr
[ULØ]  118:             cld
[K74]  119:             mov     CX,word ptr Height
[9S1]  12Ø:             xchg    CH,CL
[933]  121: ClrLoop:    push    CX
[PE1]  122:             push    DI
[6T1]  123:             xor     CH,CH
[KZ1]  124:             rep     stosw
[GC1]  125:             pop     DI
[VD1]  126:             pop     CX
[L12]  127:             add     di,ØAØh
[191]  128:             dec     CH
[FG2]  129:             jnz     ClrLoop
[GD1]  13Ø:             pop     ES
[LSØ]  131:             popf
[HM3]  132:             mov     word ptr ActY,Ø
[2S2]  133:             jmp     SetCursor
[E92]  134: ClrScr      endp
[ØØØ]  135:
[V63]  136: ClrEol      proc    near
[ZXØ]  137:             pushf
[PF1]  138:             push    ES
[KW3]  139:             mov     DX,word ptr ActY
[GG4]  14Ø:             add     DX,word ptr WinYPos
[VV2]  141:             call    BerechneXY
[6D2]  142:             mov     CL,Width
[AA2]  143:             sub     CL,ActX
[6T1]  144:             xor     CH,CH
[ZX1]  145:             mov     AL,2Øh
[KS2]  146:             mov     AH,WinAttr
[ULØ]  147:             cld
[KZ1]  148:             rep     stosw
[GD1]  149:             pop     ES
[LSØ]  15Ø:             popf
[3PØ]  151:             ret
[692]  152: ClrEol      endp
[ØØØ]  153:
[LD3]  154: DelLine     proc    near
[ZXØ]  155:             pushf
[PF1]  156:             push    ES
[ULØ]  157:             cld
[HM4]  158:             mov     DX,word ptr WinYPos
[182]  159:             add     DL,ActY
[VV2]  16Ø:             call    BerechneXY
[WB2]  161:             mov     AL,ActY
[EA1]  162:             inc     AL
[WE2]  163:             cmp     AL,Height
[DB2]  164:             jz      LastLine
[6U1]  165:             mov     SI,DI
[H22]  166:             add     SI,ØAØh
[K74]  167:             mov     CX,word ptr Height
[9S1]  168:             xchg    CH,CL
[191]  169:             dec     CH
[2A2]  17Ø:             sub     CH,ActY
[MF1]  171:             push    DS
[PF1]  172:             push    ES
[ED1]  173:             pop     DS
[Y13]  174: DelLoop:    push    CX
[MF1]  175:             push    SI
[PE1]  176:             push    DI
[6T1]  177:             xor     CH,CH
[LZ1]  178:             rep     movsw
[GC1]  179:             pop     DI
[L12]  18Ø:             add     DI,ØAØh
[ED1]  181:             pop     SI
[H22]  182:             add     SI,ØAØh
[VD1]  183:             pop     CX
[191]  184:             dec     CH
[4F2]  185:             jnz     DelLoop
[ED1]  186:             pop     DS
[2G3]  187: LastLine    label   near
[6D2]  188:             mov     CL,Width
[6T1]  189:             xor     CH,CH
[ZX1]  19Ø:             mov     AL,2Øh
[KS2]  191:             mov     AH,WinAttr
[ULØ]  192:             cld
[KZ1]  193:             rep     stosw
[GD1]  194:             pop     ES
[LSØ]  195:             popf
[3PØ]  196:             ret
[GC2]  197: DelLine     endp
[ØØØ]  198:
[MF3]  199: InsLine     proc    near
[ZXØ]  2ØØ:             pushf
[PF1]  2Ø1:             push    ES
[ULØ]  2Ø2:             cld
[HM4]  2Ø3:             mov     DX,word ptr WinYPos
[182]  2Ø4:             add     DL,ActY
[VV2]  2Ø5:             call    BerechneXY
[WB2]  2Ø6:             mov     AL,ActY
[EA1]  2Ø7:             inc     AL
[WE2]  2Ø8:             cmp     AL,Height
[DB2]  2Ø9:             jz      LastLine
[PE1]  21Ø:             push    DI
[HM4]  211:             mov     DX,word ptr WinYPos
[LC2]  212:             add     DL,Height
[F91]  213:             dec     DL
[VV2]  214:             call    BerechneXY
[6U1]  215:             mov     SI,DI
[Ø52]  216:             sub     SI,ØAØh
[K74]  217:             mov     CX,word ptr Height
[9S1]  218:             xchg    CH,CL
[191]  219:             dec     CH
[2A2]  22Ø:             sub     CH,ActY
[MF1]  221:             push    DS
[PF1]  222:             push    ES
[ED1]  223:             pop     DS
[Z33]  224: InsLoop:    push    CX
[MF1]  225:             push    SI
[PE1]  226:             push    DI
[6T1]  227:             xor     CH,CH
[LZ1]  228:             rep     movsw
[GC1]  229:             pop     DI
[242]  23Ø:             sub     DI,ØAØh
```

```
[ED1]  231:                pop     SI
[Ø52]  232:                sub     SI,ØAØh
[VD1]  233:                pop     CX
[191]  234:                dec     CH
[5H2]  235:                jnz     InsLoop
[ED1]  236:                pop     DS
[GC1]  237:                pop     DI
[6G2]  238:                jmp     LastLine
[HE2]  239: InsLine        endp
[ØØØ]  24Ø:
[8K3]  241: SetColor       proc    near
[ZX2]  242:                mov     WinAttr,AL
[3PØ]  243:                ret
[5H2]  244: SetColor       endp
[ØØØ]  245:
[GH3]  246: GetColor       proc    near
[YS2]  247:                mov     AL,WinAttr
[3PØ]  248:                ret
[DG2]  249: GetColor       endp
[ØØØ]  25Ø:
[ZH3]  251: DefWindow      proc    near
[FK1]  252:                or      AH,AH
[EV1]  253:                jnz     DefW1
[872]  254: ExitDefW:      stc
[3PØ]  255:                ret
[MU2]  256: DefW1:         or      CH,CH
[KA2]  257:                jz      ExitDefW
[9L1]  258:                or      CL,CL
[KA2]  259:                jz      ExitDefW
[NX1]  26Ø:                cmp     DH,4Fh
[782]  261:                ja      ExitDefW
[2X1]  262:                cmp     DL,18h
[782]  263:                ja      ExitDefW
[6U1]  264:                mov     DI,DX
[1N1]  265:                add     DH,CH
[NN1]  266:                add     DL,CL
[391]  267:                dec     DH
[F91]  268:                dec     DL
[NX1]  269:                cmp     DH,4Fh
[782]  27Ø:                ja      ExitDefW
[2X1]  271:                cmp     DL,18h
[782]  272:                ja      ExitDefW
[NU1]  273:                mov     DX,DI
[XG1]  274:                and     AH,7
[XF1]  275:                push    AX
[8T1]  276:                mov     BL,AH
[3T1]  277:                xor     BH,BH
[VL1]  278:                shl     BX,1
[VL1]  279:                shl     BX,1
[VL1]  28Ø:                shl     BX,1
[6W3]  281:                add     BX,offset Windows
[RD1]  282:                pop     AX
[WF2]  283:                mov     [BX],DX
[4L2]  284:                mov     [BX+2],CX
[VY3]  285:                mov     word ptr [BX+4],Ø
[KK2]  286:                mov     [BX+6],AL
[1Z3]  287:                mov     byte ptr [BX+7],Ø
[UP2]  288:                cmp     AH,WSelect
[TG2]  289:                jz      SelectNew
[PLØ]  29Ø:                clc
[3PØ]  291:                ret
[NG4]  292: SelectNew:     call    SelectWin
[PLØ]  293:                clc
[3PØ]  294:                ret
[WG2]  295: DefWindow      endp
[ØØØ]  296:
[RL3]  297: AskWindow      proc    near
[XG1]  298:                and     AH,7
[8T1]  299:                mov     BL,AH
[3T1]  3ØØ:                xor     BH,BH
[CM1]  3Ø1:                mov     CL,3
[5T1]  3Ø2:                shl     BX,CL
[6W3]  3Ø3:                add     BX,offset Windows
[RY3]  3Ø4:                mov     DX,word ptr [BX]
[Y84]  3Ø5:                mov     CX,word ptr [BX+2]
[784]  3Ø6:                mov     AL,byte ptr [BX+6]
[3PØ]  3Ø7:                ret
[MK2]  3Ø8: AskWindow      endp
[ØØØ]  3Ø9:
[NL3]  31Ø: SelWindow      proc    near
[XG1]  311:                and     AH,7
[FV2]  312:                mov     WSelect,AH
[8P2]  313:                jmp     SelectWin
[KK2]  314: SelWindow      endp
[ØØØ]  315:
[5K3]  316: PutFrame       proc    near
[ØZ3]  317:                cmp     byte ptr [SI+5],1
[S72]  318:                jz      FrameOk
[3PØ]  319:                ret
[D44]  32Ø: FrameOk:       call    SaveBack
[GØ4]  321:                mov     byte ptr [SI+5],Ø
[EY3]  322:                mov     DX,word ptr [SI]
[CS1]  323:                xchg    DH,DL
[ZXØ]  324:                pushf
[PF1]  325:                push    ES
[ULØ]  326:                cld
[VV2]  327:                call    BerechneXY
[N84]  328:                mov     CX,word ptr [SI+2]
[9S1]  329:                xchg    CH,CL
[E74]  33Ø:                mov     AH,byte ptr [SI+4]
[MF1]  331:                push    SI
```

```
[G84]  332:                mov     SI,word ptr [SI+6]
[XZ2]  333: FrLoop:        push    CX
[PE1]  334:                push    DI
[6T1]  335:                xor     CH,CH
[HF2]  336: FrLoop1:       lodsb
[3L1]  337:                or      AL,AL
[592]  338:                jz      Special1
[MK1]  339:                cmp     AL,1
[692]  34Ø:                jz      Special2
[NK1]  341:                cmp     AL,2
[792]  342:                jz      Special3
[ZYØ]  343:                stosw
[CW3]  344: ContFr:        loop    FrLoop1
[GC1]  345:                pop     DI
[L12]  346:                add     DI,ØAØh
[VD1]  347:                pop     CX
[191]  348:                dec     CH
[N62]  349:                jnz     FrLoop
[ED1]  35Ø:                pop     SI
[GD1]  351:                pop     ES
[LSØ]  352:                popf
[3PØ]  353:                ret
[VH2]  354: Special1:      lodsb
[2T1]  355:                mov     AH,AL
[EB2]  356:                jmp     FrLoop1
[F33]  357: Special2:      add     DI,2
[C43]  358:                jmp     short ContFr
[AX5]  359: Special3:      and     byte ptr ES:[DI+1],7
[HD3]  36Ø:                jmp     short Special2
[2H2]  361: PutFrame       endp
[ØØØ]  362:
[XE3]  363: DelFrame       proc    near
[YY3]  364:                cmp     byte ptr [SI+5],Ø
[C82]  365:                jz      DelFrOk
[3PØ]  366:                ret
[TG5]  367: DelFrOk:       mov     byte ptr [SI+5],1
[EY3]  368:                mov     DX,word ptr [SI]
[CS1]  369:                xchg    DH,DL
[ZXØ]  37Ø:                pushf
[PF1]  371:                push    ES
[ULØ]  372:                cld
[VV2]  373:                call    BerechneXY
[N84]  374:                mov     CX,word ptr [SI+2]
[9S1]  375:                xchg    CH,CL
[M84]  376:                mov     SI,word ptr [SI+8]
[M73]  377: DelFrLoop:     push    CX
[PE1]  378:                push    DI
[6T1]  379:                xor     CH,CH
[LZ1]  38Ø:                rep     movsw
[GC1]  381:                pop     DI
[L12]  382:                add     DI,ØAØh
[VD1]  383:                pop     CX
[191]  384:                dec     CH
[6P2]  385:                jnz     DelFrLoop
[GD1]  386:                pop     ES
[LSØ]  387:                popf
[3PØ]  388:                ret
[UD2]  389: DelFrame       endp
[ØØØ]  39Ø:
[PK3]  391: ScrollUp       proc    near
[ZXØ]  392:                pushf
[PF1]  393:                push    ES
[HM4]  394:                mov     DX,word ptr WinYPos
[VV2]  395:                call    BerechneXY
[4A2]  396:                cmp     Height,1
[BG2]  397:                jnz     ScrUpOk
[6G2]  398:                jmp     LastLine
[V54]  399: ScrUpOk:       inc     ScrUpFlag
[K74]  4ØØ:                mov     CX,word ptr Height
[9S1]  4Ø1:                xchg    CH,CL
[191]  4Ø2:                dec     CH
[6U1]  4Ø3:                mov     SI,DI
[H22]  4Ø4:                add     SI,ØAØh
[MF1]  4Ø5:                push    DS
[PF1]  4Ø6:                push    ES
[ED1]  4Ø7:                pop     DS
[F93]  4Ø8: ScrUpLoop:     push    CX
[MF1]  4Ø9:                push    SI
[PE1]  41Ø:                push    DI
[6T1]  411:                xor     CH,CH
[LZ1]  412:                rep     movsw
[GC1]  413:                pop     DI
[L12]  414:                add     DI,ØAØh
[ED1]  415:                pop     SI
[H22]  416:                add     SI,ØAØh
[VD1]  417:                pop     CX
[191]  418:                dec     CH
[1S2]  419:                jnz     ScrUpLoop
[ED1]  42Ø:                pop     DS
[6G2]  421:                jmp     LastLine
[LH2]  422: ScrollUp       endp
[ØØØ]  423:
[GG3]  424: ScrollDn       proc    near
[ZXØ]  425:                pushf
[PF1]  426:                push    ES
[HM4]  427:                mov     DX,word ptr WinYPos
[LC2]  428:                add     DL,Height
[F91]  429:                dec     DL
```

Listing 1. »system.asm« stellt Ihnen 40 nützliche Routinen zur Verfügung

```
[VV2]  43Ø:             call    BerechneXY
[4A2]  431:             cmp     Height,1
[PF2]  432:             jnz     ScrDnOk
[6G2]  433:             jmp     LastLine
[GF5]  434: ScrDnOk:    mov     CX,word ptr Height
[9S1]  435:             xchg    CH,CL
[191]  436:             dec     CH
[6U1]  437:             mov     SI,DI
[Ø52]  438:             sub     SI,ØAØh
[MF1]  439:             push    DS
[PF1]  44Ø:             push    ES
[ED1]  441:             pop     DS
[U83]  442: ScrDnLoop:  push    CX
[MF1]  443:             push    SI
[PE1]  444:             push    DI
[6T1]  445:             xor     CH,CH
[LZ1]  446:             rep     movsw
[GC1]  447:             pop     DI
[242]  448:             sub     DI,ØAØh
[ED1]  449:             pop     SI
[Ø52]  45Ø:             sub     SI,ØAØh
[VD1]  451:             pop     CX
[191]  452:             dec     CH
[CR2]  453:             jnz     ScrDnLoop
[ED1]  454:             pop     DS
[6G2]  455:             jmp     LastLine
[DF2]  456: ScrollDn    endp
[ØØØ]  457:
[8H3]  458: ScrollLf    proc    near
[ZXØ]  459:             pushf
[PF1]  46Ø:             push    ES
[HM4]  461:             mov     DX,word ptr WinYPos
[VV2]  462:             call    BerechneXY
[K62]  463:             cmp     Width,1
[EF2]  464:             jnz     ScrLfOk
[E93]  465: ClrCol      label   near
[KG2]  466:             mov     CL,Height
[6T1]  467:             xor     CH,CH
[ZX1]  468:             mov     AL,2Øh
[KS2]  469:             mov     AH,WinAttr
[5T2]  47Ø: ClColLoop:  stosw
[AV1]  471:             add     DI,7Eh
[XN2]  472:             loop    ClColLoop
[GD1]  473:             pop     ES
[LSØ]  474:             popf
[3PØ]  475:             ret
[HB3]  476: ScrLfOk:    mov     SI,DI
[1H1]  477:             add     SI,2
[K74]  478:             mov     CX,word ptr Height
[9S1]  479:             xchg    CH,CL
[D91]  48Ø:             dec     CL
[ZX1]  481:             mov     AL,2Øh
[KS2]  482:             mov     AH,WinAttr
[MF1]  483:             push    DS
[PF1]  484:             push    ES
[ED1]  485:             pop     DS
[K83]  486: ScrLfLoop:  push    CX
[MF1]  487:             push    SI
[PE1]  488:             push    DI
[6T1]  489:             xor     CH,CH
[LZ1]  49Ø:             rep     movsw
[ZYØ]  491:             stosw
[GC1]  492:             pop     DI
[L12]  493:             add     DI,ØAØh
[ED1]  494:             pop     SI
[H22]  495:             add     SI,ØAØh
[VD1]  496:             pop     CX
[191]  497:             dec     CH
[4R2]  498:             jnz     ScrLfLoop
[ED1]  499:             pop     DS
[GD1]  5ØØ:             pop     ES
[LSØ]  5Ø1:             popf
[3PØ]  5Ø2:             ret
[5G2]  5Ø3: ScrollLf    endp
[ØØØ]  5Ø4:
[1K3]  5Ø5: ScrollRg    proc    near
[ZXØ]  5Ø6:             pushf
[PF1]  5Ø7:             push    ES
[HM4]  5Ø8:             mov     DX,word ptr WinYPos
[VV2]  5Ø9:             call    BerechneXY
[K62]  51Ø:             cmp     Width,1
[PF2]  511:             jnz     ScrRgOk
[W33]  512:             jmp     short ClrCol
[X24]  513: ScrRgOk:    mov     AL,Width
[991]  514:             dec     AL
[ØT1]  515:             xor     AH,AH
[TL1]  516:             shl     AX,1
[2P1]  517:             add     DI,AX
[6U1]  518:             mov     SI,DI
[GL1]  519:             sub     SI,2
[K74]  52Ø:             mov     CX,word ptr Height
[9S1]  521:             xchg    CH,CL
[D91]  522:             dec     CL
[ZX1]  523:             mov     AL,2Øh
[KS2]  524:             mov     AH,WinAttr
[MF1]  525:             push    DS
[PF1]  526:             push    ES
[ED1]  527:             pop     DS
[KNØ]  528:             std
[U83]  529: ScrRgLoop:  push    CX
[MF1]  53Ø:             push    SI
```

```
[PE1]  531:             push    DI
[6T1]  532:             xor     CH,CH
[LZ1]  533:             rep     movsw
[ZYØ]  534:             stosw
[GC1]  535:             pop     DI
[L12]  536:             add     DI,ØAØh
[ED1]  537:             pop     SI
[H22]  538:             add     SI,ØAØh
[VD1]  539:             pop     CX
[191]  54Ø:             dec     CH
[CR2]  541:             jnz     ScrRgLoop
[ED1]  542:             pop     DS
[GD1]  543:             pop     ES
[LSØ]  544:             popf
[3PØ]  545:             ret
[YG2]  546: ScrollRg    endp
[ØØØ]  547:
[5D3]  548: GotoXY      proc    near
[TT3]  549:             mov     word ptr ActY,DX
[2S2]  55Ø:             jmp     SetCursor
[2C2]  551: GotoXY      endp
[ØØØ]  552:
[773]  553: GetXY       proc    near
[KW3]  554:             mov     DX,word ptr ActY
[3PØ]  555:             ret
[462]  556: GetXY       endp
[ØØØ]  557:
[1L3]  558: ShowCurs    proc    near
[NK2]  559:             mov     CurFlag,Ø
[2S2]  56Ø:             jmp     SetCursor
[YH2]  561: ShowCurs    endp
[ØØØ]  562:
[GG3]  563: HideCurs    proc    near
[RK2]  564:             mov     CurFlag,1
[2S2]  565:             jmp     SetCursor
[DF2]  566: HideCurs    endp
[ØØØ]  567:
[Ø73]  568: Print       proc    near
[ZXØ]  569:             pushf
[MF1]  57Ø:             push    SI
[PF1]  571:             push    ES
[KW3]  572:             mov     DX,word ptr ActY
[3G1]  573:             push    DX
[GG4]  574:             add     DX,word ptr WinYPos
[VV2]  575:             call    BerechneXY
[XD1]  576:             pop     DX
[9B1]  577:             inc     SI
[CVØ]  578:             lodsb
[3L1]  579:             or      AL,AL
[WH2]  58Ø:             jz      ExitPrint
[KT1]  581:             mov     CL,AL
[6T1]  582:             xor     CH,CH
[KS2]  583:             mov     AH,WinAttr
[VM2]  584: PrtLoop:    lodsb
[LX1]  585:             cmp     AL,ØDh
[Y71]  586:             jz      cr
[DX1]  587:             cmp     AL,ØAh
[C71]  588:             jz      lf
[ZYØ]  589:             stosw
[8A1]  59Ø:             inc     DH
[AB2]  591:             cmp     DH,Width
[5S1]  592:             jb      PrtOk
[9T1]  593:             xor     DH,DH
[MA1]  594:             inc     DL
[2F2]  595:             cmp     DL,Height
[Z82]  596:             jb      NewKoor
[P13]  597: PrtScroll:  dec     DL
[XF1]  598:             push    AX
[1G1]  599:             push    CX
[3G1]  6ØØ:             push    DX
[MF1]  6Ø1:             push    SI
[BH2]  6Ø2:             call    ScrollUp
[ED1]  6Ø3:             pop     SI
[XD1]  6Ø4:             pop     DX
[VD1]  6Ø5:             pop     CX
[RD1]  6Ø6:             pop     AX
[643]  6Ø7: NewKoor:    push    DX
[GG4]  6Ø8:             add     DX,word ptr WinYPos
[VV2]  6Ø9:             call    BerechneXY
[XD1]  61Ø:             pop     DX
[DW3]  611: PrtOk:      loop    PrtLoop
[TT3]  612:             mov     word ptr ActY,DX
[CR2]  613:             call    SetCursor
[H33]  614: ExitPrint:  pop     ES
[ED1]  615:             pop     SI
[LSØ]  616:             popf
[3PØ]  617:             ret
[ØU2]  618: cr:         xor     DH,DH
[ØG2]  619:             jmp     NewKoor
[W82]  62Ø: lf:         inc     DL
[2F2]  621:             cmp     DL,Height
[Z82]  622:             jb      NewKoor
[GR2]  623:             jmp     PrtScroll
[X52]  624: Print       endp
[ØØØ]  625:
[NT3]  626: ConvString  proc    near
[ZXØ]  627:             pushf
[MF1]  628:             push    SI
[PE1]  629:             push    DI
[ULØ]  63Ø:             cld
[ØT1]  631:             xor     AH,AH
```

```
[MF1]  632:              push    SI
[XL2]  633: CnvLoop:     lodsb
[3L1]  634:              or      AL,AL
[FB2]  635:              jz      LengthOk
[2A1]  636:              inc     AH
[4A3]  637:              jmp     short CnvLoop
[4Y2]  638: LengthOk:    pop     SI
[AG2]  639:              mov     [DI+1],AH
[FK1]  64Ø:              or      AH,AH
[9D2]  641:              jz      NoString
[9A2]  642:              cmp     AH,[DI]
[TF2]  643:              jbe     StringOk
[VB2]  644:              mov     AH,[DI]
[RE3]  645: StringOk:    mov     CL,AH
[6T1]  646:              xor     CH,CH
[AG2]  647:              mov     [DI+1],AH
[3G1]  648:              add     DI,2
[9Y1]  649:              rep     movsb
[FZ2]  65Ø: NoString:    pop     DI
[ED1]  651:              pop     SI
[LSØ]  652:              popf
[3PØ]  653:              ret
[KS2]  654: ConvString   endp
[ØØØ]  655:
[ME3]  656: ReadKey      proc    near
[RU2]  657:              mov     KeyFlag,AL
[FV2]  658:              mov     OutFlag,AH
[783]  659: RKLoop:      xor     AH,AH
[GK1]  66Ø:              int     16h
[ZG2]  661:              cmp     KeyFlag,Ø
[NX1]  662:              jz      CharOk
[DT1]  663:              cmp     AL,"a"
[DV1]  664:              jb      CharOk
[ZU1]  665:              cmp     AL,"z"
[AV1]  666:              ja      CharOk
[W42]  667:              and     AL,ØDFh
[923]  668: CharOk:      or      CX,CX
[NG2]  669:              jnz     TstChar
[3PØ]  67Ø:              ret
[2M2]  671: TstChar:     pushf
[1G1]  672:              push    CX
[PF1]  673:              push    ES
[RT1]  674:              mov     DI,SI
[MF1]  675:              push    DS
[GD1]  676:              pop     ES
[ULØ]  677:              cld
[ZZ1]  678:              repnz   scasb
[BK2]  679:              jnz     ChrFound
[GD1]  68Ø:              pop     ES
[VD1]  681:              pop     CX
[LSØ]  682:              popf
[L63]  683:              jmp     short RKLoop
[X43]  684: ChrFound:    push    AX
[3K2]  685:              cmp     OutFlag,Ø
[WE2]  686:              jz      NoOutput
[ZF1]  687:              push    BX
[MF1]  688:              push    SI
[P23]  689:              mov     SysStr+2,AL
[WW3]  69Ø:              mov     SI,offset SysStr
[NX1]  691:              call    Print
[ED1]  692:              pop     SI
[TD1]  693:              pop     BX
[X13]  694: NoOutput:    pop     AX
[GD1]  695:              pop     ES
[VD1]  696:              pop     CX
[LSØ]  697:              popf
[3PØ]  698:              ret
[HD2]  699: ReadKey      endp
[ØØØ]  7ØØ:
[R63]  7Ø1: Flush        proc    near
[EMØ]  7Ø2:              cli
[PF1]  7Ø3:              push    ES
[GV1]  7Ø4:              xor     AX,AX
[3V1]  7Ø5:              mov     ES,AX
[F53]  7Ø6:              mov     AX,ES:[Ø41Ah]
[G23]  7Ø7:              mov     ES:[Ø41Ch],AX
[GD1]  7Ø8:              pop     ES
[6PØ]  7Ø9:              sti
[3PØ]  71Ø:              ret
[M52]  711: Flush        endp
[ØØØ]  712:
[973]  713: Input        proc    near
[ZXØ]  714:              pushf
[1G1]  715:              push    CX
[MF1]  716:              push    SI
[PE1]  717:              push    DI
[7F1]  718:              push    BP
[PF1]  719:              push    ES
[MF1]  72Ø:              push    SI
[MF1]  721:              push    DS
[GD1]  722:              pop     ES
[ØØ3]  723:              mov     ScrUpFlag,Ø
[9S2]  724:              mov     DelFlag,AH
[RU2]  725:              mov     KeyFlag,AL
[TZ2]  726:              mov     AnzKeys,CX
[KW3]  727:              mov     DX,word ptr ActY
[EG2]  728:              mov     PosXY,DX
[ZG2]  729:              mov     Bufadr,DI
[VL1]  73Ø:              mov     AH,2
[GK1]  731:              int     16h
[2V1]  732:              and     AL,8Øh
```

```
[NL2]  733:              mov     Insert,AL
[WT3]  734:              mov     DI,offset StrBuf
[PE1]  735:              push    DI
[YX1]  736:              mov     CX,257
[ULØ]  737:              cld
[9Y1]  738:              rep     movsb
[ED1]  739:              pop     SI
[NX1]  74Ø:              call    Print
[EY2]  741:              cmp     ScrUpFlag,Ø
[7D2]  742:              jz      NoScroll
[ZG2]  743:              mov     DX,PosXY
[R23]  744:              sub     DL,ScrUpFlag
[EG2]  745:              mov     PosXY,DX
[ØØ3]  746:              mov     ScrUpFlag,Ø
[1H3]  747: NoScroll:    mov     BP,SI
[XY2]  748:              mov     AL,DS:[BP+1]
[AK2]  749:              mov     BufPos,AL
[ØT1]  75Ø:              xor     AH,AH
[AH1]  751:              add     AX,2
[LP1]  752:              add     BP,AX
[283]  753: GetKey:      xor     AH,AH
[GK1]  754:              int     16h
[ZG2]  755:              cmp     KeyFlag,Ø
[NA2]  756:              jz      NoUpChr
[DT1]  757:              cmp     AL,"a"
[D82]  758:              jb      NoUpChr
[ZU1]  759:              cmp     AL,"z"
[A82]  76Ø:              ja      NoUpChr
[W42]  761:              and     AL,ØDFh
[154]  762: NoUpChr:     cmp     AnzKeys,Ø
[GT1]  763:              jz      KeyOk
[WF2]  764:              mov     DI,BufAdr
[4W2]  765:              mov     CX,AnzKeys
[ZZ1]  766:              repnz   scasb
[MZ1]  767:              jnz     KeyOk
[643]  768:              jmp     short GetKey
[W43]  769: KeyOk:       cmp     AL,1Bh
[MB2]  77Ø:              jnz     ChkKey1
[4R1]  771:              mov     AL,-1
[ED1]  772:              pop     SI
[U33]  773: ExitInput:   pop     ES
[1D1]  774:              pop     BP
[GC1]  775:              pop     DI
[ED1]  776:              pop     SI
[VD1]  777:              pop     CX
[LSØ]  778:              popf
[3PØ]  779:              ret
[683]  78Ø: ChkKey1:     cmp     AL,ØDh
[SB2]  781:              jnz     ChkKey2
[CE2]  782:              mov     CL,[SI+1]
[6T1]  783:              xor     CH,CH
[EH1]  784:              add     CX,2
[GC1]  785:              pop     DI
[PE1]  786:              push    DI
[9Y1]  787:              rep     movsb
[ZG2]  788:              mov     DX,PosXY
[TT3]  789:              mov     word ptr ActY,DX
[ED1]  79Ø:              pop     SI
[NX1]  791:              call    Print
[MT1]  792:              xor     AL,AL
[AW3]  793:              jmp     short ExitInput
[Ø43]  794: ChkKey2:     cmp     AH,82
[WB2]  795:              jnz     ChkKey3
[XF4]  796:              xor     byte ptr Insert,8Øh
[852]  797:              jmp     GetKey
[343]  798: ChkKey3:     cmp     AH,75
[ØC2]  799:              jnz     ChkKey4
[ØF2]  8ØØ:              jmp     CurLeft
[943]  8Ø1: ChkKey4:     cmp     AH,77
[4C2]  8Ø2:              jnz     ChkKey5
[NG2]  8Ø3:              jmp     CurRight
[843]  8Ø4: ChkKey5:     cmp     AH,72
[8C2]  8Ø5:              jnz     ChkKey6
[XY1]  8Ø6:              jmp     CurUp
[E43]  8Ø7: ChkKey6:     cmp     AH,8Ø
[CC2]  8Ø8:              jnz     ChkKey7
[7G2]  8Ø9:              jmp     CurDown
[F43]  81Ø: ChkKey7:     cmp     AH,71
[GC2]  811:              jnz     ChkKey8
[9E2]  812:              jmp     CurHome
[V43]  813: ChkKey8:     cmp     AH,79
[MC2]  814:              jnz     ChkKey9
[N42]  815:              jmp     CurEnd
[X43]  816: ChkKey9:     cmp     AH,83
[4D2]  817:              jnz     ChkKey1Ø
[U42]  818:              jmp     CurDel
[1R3]  819: ChkKey1Ø:    cmp     AX,ØE Ø8h
[5D2]  82Ø:              jnz     ChkKey11
[GG2]  821:              jmp     CurDelLf
[SP3]  822: ChkKey11:    cmp     AX,77ØØh
[6D2]  823:              jnz     ChkKey12
[Z22]  824:              jmp     DelAll
[693]  825: ChkKey12:    cmp     AL,2Øh
[PU1]  826:              jae     ChrOk
[852]  827:              jmp     GetKey
[LP3]  828: ChrOk:       cmp     DelFlag,Ø
[ØX1]  829:              jz      ChrOk1
[FG2]  83Ø:              mov     DelFlag,Ø
```

Listing 1. »system.asm« stellt Ihnen 40 nützliche Routinen zur Verfügung

```
[XF1]  831:              push    AX
[1U2]  832:              call    FreeBuffer
[RD1]  833:              pop     AX
[TX3]  834: ChrOk1:      mov     BL,BufPos
[NB2]  835:              cmp     BL,[SI]
[Ø52]  836:              jnz     FreeOk
[MF1]  837:              push    SI
[PE1]  838:              push    DI
[7F1]  839:              push    BP
[PF1]  84Ø:              push    ES
[772]  841:              mov     AX,ØEØ7h
[YH1]  842:              int     1Øh
[GD1]  843:              pop     ES
[1D1]  844:              pop     BP
[GC1]  845:              pop     DI
[ED1]  846:              pop     SI
[852]  847:              jmp     GetKey
[K64]  848: FreeOk:      call    ChkInsert
[XV2]  849:              mov     DS:[BP],AL
[8E2]  85Ø:              mov     AL,[SI+1]
[LB2]  851:              cmp     AL,[SI]
[B82]  852:              jz      DontInc
[1T3]  853:              inc     byte ptr [SI+1]
[954]  854: DontInc:     mov     DX,PosXY
[EY2]  855:              cmp     ScrUpFlag,Ø
[BA2]  856:              jz      PrintOk
[F91]  857:              dec     DL
[EG2]  858:              mov     PosXY,DX
[ØØ3]  859:              mov     ScrUpFlag,Ø
[XE5]  86Ø: PrintOk:     xchg    DX,word ptr ActY
[3G1]  861:              push    DX
[NX1]  862:              call    Print
[V93]  863:              pop     word ptr ActY
[ZG2]  864:              mov     DX,PosXY
[R23]  865:              sub     DL,ScrUpFlag
[EG2]  866:              mov     PosXY,DX
[ØØ3]  867:              mov     ScrUpFlag,Ø
[NG2]  868:              jmp     CurRight
[DV5]  869: PosCursor:   mov     word ptr ActY,DX
[MF1]  87Ø:              push    SI
[PE1]  871:              push    DI
[7F1]  872:              push    BP
[CR2]  873:              call    SetCursor
[1D1]  874:              pop     BP
[GC1]  875:              pop     DI
[ED1]  876:              pop     SI
[852]  877:              jmp     GetKey
[TØ4]  878: CurLeft:     mov     DelFlag,Ø
[FB2]  879:              cmp     BufPos,Ø
[4G2]  88Ø:              jnz     CurLfOk
[852]  881:              jmp     GetKey
[HU2]  882: CurLfOk:     dec     BP
[522]  883:              dec     BufPos
[KW3]  884:              mov     DX,word ptr ActY
[EK1]  885:              sub     DH,1
[5T2]  886:              jns     PosCursor
[WC2]  887:              mov     DH,Width
[391]  888:              dec     DH
[F91]  889:              dec     DL
[PS2]  89Ø:              jmp     PosCursor
[824]  891: CurRight:    mov     DelFlag,Ø
[8K2]  892:              mov     AL,BufPos
[PC2]  893:              cmp     AL,[SI+1]
[CG2]  894:              jnz     CurRgOk
[852]  895:              jmp     GetKey
[GV2]  896: CurRgOk:     inc     BP
[A32]  897:              inc     BufPos
[KW3]  898:              mov     DX,word ptr ActY
[8A1]  899:              inc     DH
[AB2]  9ØØ:              cmp     DH,Width
[NH2]  9Ø1:              jb      PosCursor
[9T1]  9Ø2:              xor     DH,DH
[MA1]  9Ø3:              inc     DL
[PS2]  9Ø4:              jmp     PosCursor
[AT3]  9Ø5: CurUp:       mov     DelFlag,Ø
[T42]  9Ø6:              call    GetPos
[3L1]  9Ø7:              or      AL,AL
[LG2]  9Ø8:              jz      NoCurMove
[F91]  9Ø9:              dec     DL
[2D2]  91Ø:              mov     AL,Width
[SG2]  911:              sub     BufPos,AL
[ØT1]  912:              xor     AH,AH
[2T1]  913:              sub     BP,AX
[PS2]  914:              jmp     PosCursor
[3U3]  915: NoCurMove:   jmp     GetKey
[Ø24]  916: CurDown:     mov     DelFlag,Ø
[8K2]  917:              mov     AL,BufPos
[192]  918:              add     AL,Width
[EE2]  919:              jc      NoCurMove
[PC2]  92Ø:              cmp     AL,[SI+1]
[8E2]  921:              ja      NoCurMove
[T42]  922:              call    GetPos
[MA1]  923:              inc     DL
[2D2]  924:              mov     AL,Width
[9E2]  925:              add     BufPos,AL
[ØT1]  926:              xor     AH,AH
[LP1]  927:              add     BP,AX
[PS2]  928:              jmp     PosCursor
[2Ø4]  929: CurHome:     mov     DelFlag,Ø
[8U1]  93Ø:              mov     BP,SI
[MG1]  931:              add     BP,2
```

```
[1D2]  932:              mov     BufPos,Ø
[T42]  933:              call    GetPos
[PS2]  934:              jmp     PosCursor
[WW3]  935: CurEnd:      mov     DelFlag,Ø
[8E2]  936:              mov     AL,[SI+1]
[AK2]  937:              mov     bufpos,AL
[ØT1]  938:              xor     AH,AH
[MU1]  939:              mov     BP,AX
[7P1]  94Ø:              add     BP,SI
[MG1]  941:              add     BP,2
[T42]  942:              call    GetPos
[PS2]  943:              jmp     PosCursor
[1X3]  944: CurDel:      mov     DelFlag,Ø
[KY3]  945:              cmp     byte ptr [SI+1],Ø
[KX1]  946:              jnz     DelOk
[852]  947:              jmp     GetKey
[T13]  948: DelOk:       mov     DI,BP
[CE2]  949:              mov     CL,[SI+1]
[ZXØ]  95Ø:              pushf
[MF1]  951:              push    SI
[6U1]  952:              mov     SI,DI
[9B1]  953:              inc     SI
[ULØ]  954:              cld
[UG2]  955:              sub     CL,BufPos
[WA2]  956:              jz      NoDelete
[PF1]  957:              push    ES
[MF1]  958:              push    DS
[GD1]  959:              pop     ES
[6T1]  96Ø:              xor     CH,CH
[D91]  961:              dec     CL
[YD2]  962:              jz      NoDelete1
[9Y1]  963:              rep     movsb
[WF3]  964: NoDelete1:   mov     AL,2Øh
[NXØ]  965:              stosb
[GD1]  966:              pop     ES
[ED1]  967:              pop     SI
[MF1]  968:              push    SI
[ZG2]  969:              mov     DX,PosXY
[LV3]  97Ø:              xchg    DX,word ptr ActY
[3G1]  971:              push    DX
[NX1]  972:              call    Print
[XD1]  973:              pop     DX
[TT3]  974:              mov     word ptr ActY,DX
[ED1]  975:              pop     SI
[WR3]  976:              dec     byte ptr [SI+1]
[LSØ]  977:              popf
[PS2]  978:              jmp     PosCursor
[HX2]  979: NoDelete:    pop     SI
[LSØ]  98Ø:              popf
[852]  981:              jmp     GetKey
[324]  982: CurDelLf:    mov     DelFlag,Ø
[FB2]  983:              cmp     BufPos,Ø
[8K2]  984:              jnz     CurDLfOk
[852]  985:              jmp     GetKey
[FW2]  986: CurDLfOk:    dec     BP
[522]  987:              dec     BufPos
[KW3]  988:              mov     DX,word ptr ActY
[EK1]  989:              sub     DH,1
[KH2]  99Ø:              jns     PosCurs
[WC2]  991:              mov     DH,Width
[391]  992:              dec     DH
[F91]  993:              dec     DL
[6N5]  994: PosCurs:     mov     word ptr ActY,DX
[MF1]  995:              push    SI
[CR2]  996:              call    SetCursor
[ED1]  997:              pop     SI
[U42]  998:              jmp     CurDel
[6V3]  999: DelAll:      mov     DelFlag,Ø
[1U2] 1ØØØ:              call    FreeBuffer
[852] 1ØØ1:              jmp     GetKey
[662] 1ØØ2: Input        endp
[ØØØ] 1ØØ3:
[RL3] 1ØØ4: DeleteStr    proc    near
[CE2] 1ØØ5:              mov     CL,[SI+1]
[9L1] 1ØØ6:              or      CL,CL
[UK2] 1ØØ7:              jz      ExtDelStr
[KR1] 1ØØ8:              cmp     AH,CL
[FG2] 1ØØ9:              ja      ExtDelStr
[FK1] 1Ø1Ø:              or      AH,AH
[UK2] 1Ø11:              jz      ExtDelStr
[3L1] 1Ø12:              or      AL,AL
[MG2] 1Ø13:              jnz     DelLenOk
[GT1] 1Ø14:              mov     AL,CL
[MR1] 1Ø15:              sub     AL,AH
[EA1] 1Ø16:              inc     AL
[KB3] 1Ø17: DelLenOk:    mov     CH,AH
[5N1] 1Ø18:              add     CH,AL
[191] 1Ø19:              dec     CH
[PR1] 1Ø2Ø:              cmp     CH,CL
[FG2] 1Ø21:              ja      ExtDelStr
[DK2] 1Ø22:              jnz     OkDelStr
[DG2] 1Ø23:              sub     [SI+1],AL
[YE2] 1Ø24: ExtDelStr:   ret
[MN2] 1Ø25: OkDelStr:    pushf
[XF1] 1Ø26:              push    AX
[MF1] 1Ø27:              push    SI
[PF1] 1Ø28:              push    ES
[MF1] 1Ø29:              push    DS
[GD1] 1Ø3Ø:              pop     ES
[AE2] 1Ø31:              mov     BL,[SI+1]
[DG2] 1Ø32:              sub     [SI+1],AL
```

```
[RT1] 1Ø33:             mov     DI,SI
[CT1] 1Ø34:             mov     DL,AH
[9T1] 1Ø35:             xor     DH,DH
[5P1] 1Ø36:             add     DI,DX
[BA1] 1Ø37:             inc     DI
[6U1] 1Ø38:             mov     SI,DI
[MT1] 1Ø39:             mov     DL,AL
[3R1] 1Ø4Ø:             add     SI,DX
[3T1] 1Ø41:             xor     BH,BH
[5N1] 1Ø42:             add     AL,AH
[YR1] 1Ø43:             sub     BL,AL
[GA1] 1Ø44:             inc     BL
[FV1] 1Ø45:             mov     CX,BX
[ULØ] 1Ø46:             cld
[9Y1] 1Ø47:             rep     movsb
[GD1] 1Ø48:             pop     ES
[ED1] 1Ø49:             pop     SI
[RD1] 1Ø5Ø:             pop     AX
[LSØ] 1Ø51:             popf
[3PØ] 1Ø52:             ret
[MK2] 1Ø53: DeleteStr   endp
[ØØØ] 1Ø54:
[1R3] 1Ø55: InsertStr   proc    near
[8E2] 1Ø56:             mov     AL,[SI+1]
[3L1] 1Ø57:             or      AL,AL
[6M2] 1Ø58:             jz      ExtInsStr
[FK1] 1Ø59:             or      AH,AH
[6M2] 1Ø6Ø:             jz      ExtInsStr
[CB2] 1Ø61:             cmp     AH,[DI+1]
[VH2] 1Ø62:             ja      ExtInsStr
[992] 1Ø63:             add     AL,[DI+1]
[NA2] 1Ø64:             cmp     AL,[DI]
[VH2] 1Ø65:             ja      ExtInsStr
[ZXØ] 1Ø66:             pushf
[XF1] 1Ø67:             push    AX
[MF1] 1Ø68:             push    SI
[PE1] 1Ø69:             push    DI
[PF1] 1Ø7Ø:             push    ES
[MF1] 1Ø71:             push    DS
[GD1] 1Ø72:             pop     ES
[ED2] 1Ø73:             mov     CL,[DI+1]
[SR1] 1Ø74:             sub     CL,AH
[KA1] 1Ø75:             inc     CL
[6T1] 1Ø76:             xor     CH,CH
[AE2] 1Ø77:             mov     BL,[SI+1]
[MF1] 1Ø78:             push    SI
[PE1] 1Ø79:             push    DI
[6U1] 1Ø8Ø:             mov     SI,DI
[GD2] 1Ø81:             mov     DL,[DI+1]
[9T1] 1Ø82:             xor     DH,DH
[3R1] 1Ø83:             add     SI,DX
[9B1] 1Ø84:             inc     SI
[3T1] 1Ø85:             xor     BH,BH
[RT1] 1Ø86:             mov     DI,SI
[3P1] 1Ø87:             add     DI,BX
[KNØ] 1Ø88:             std
[9Y1] 1Ø89:             rep     movsb
[GC1] 1Ø9Ø:             pop     DI
[ED1] 1Ø91:             pop     SI
[1H1] 1Ø92:             add     SI,2
[ME2] 1Ø93:             mov     CL,[SI-1]
[6T1] 1Ø94:             xor     CH,CH
[CT1] 1Ø95:             mov     DL,AH
[5P1] 1Ø96:             add     DI,DX
[BA1] 1Ø97:             inc     DI
[ULØ] 1Ø98:             cld
[9Y1] 1Ø99:             rep     movsb
[GD1] 11ØØ:             pop     ES
[GC1] 11Ø1:             pop     DI
[ED1] 11Ø2:             pop     SI
[8E2] 11Ø3:             mov     AL,[SI+1]
[HC2] 11Ø4:             add     [DI+1],AL
[RD1] 11Ø5:             pop     AX
[LSØ] 11Ø6:             popf
[AG2] 11Ø7: ExtInsStr:  ret
[YN2] 11Ø8: InsertStr   endp
[ØØØ] 11Ø9:
[6K3] 111Ø: CopyStr     proc    near
[8Y3] 1111:             mov     byte ptr [DI+1],Ø
[CE2] 1112:             mov     CL,[SI+1]
[9L1] 1113:             or      CL,CL
[TL2] 1114:             jz      ExtCopStr
[FK1] 1115:             or      AH,AH
[TL2] 1116:             jz      ExtCopStr
[KR1] 1117:             cmp     AH,CL
[EH2] 1118:             ja      ExtCopStr
[3L1] 1119:             or      AL,AL
[H42] 112Ø:             jnz     CLenOk
[GT1] 1121:             mov     AL,CL
[MR1] 1122:             sub     AL,AH
[EA1] 1123:             inc     AL
[PN3] 1124: CLenOk:     cmp     AL,[DI]
[4D2] 1125:             jbe     CountOk
[7C2] 1126:             mov     AL,[DI]
[ØC3] 1127: CountOk:    mov     CH,AH
[5N1] 1128:             add     CH,AL
[191] 1129:             dec     CH
[PR1] 113Ø:             cmp     CH,CL
[EH2] 1131:             ja      ExtCopStr
[ZXØ] 1132:             pushf
```

```
[XF1] 1133:             push    AX
[MF1] 1134:             push    SI
[PE1] 1135:             push    DI
[PF1] 1136:             push    ES
[MF1] 1137:             push    DS
[GD1] 1138:             pop     ES
[KG2] 1139:             mov     [DI+1],AL
[3G1] 114Ø:             add     DI,2
[KT1] 1141:             mov     CL,AL
[6T1] 1142:             xor     CH,CH
[6T1] 1143:             mov     AL,AH
[ØT1] 1144:             xor     AH,AH
[ØR1] 1145:             add     SI,AX
[9B1] 1146:             inc     SI
[ULØ] 1147:             cld
[9Y1] 1148:             rep     movsb
[GD1] 1149:             pop     ES
[GC1] 115Ø:             pop     DI
[ED1] 1151:             pop     SI
[RD1] 1152:             pop     AX
[LSØ] 1153:             popf
[XF2] 1154: ExtCopStr:  ret
[3H2] 1155: CopyStr     endp
[ØØØ] 1156:
[ØN3] 1157: ConcatStr   proc    near
[CE2] 1158:             mov     CL,[SI+1]
[9L1] 1159:             or      CL,CL
[LL2] 116Ø:             jz      ExtConStr
[GT1] 1161:             mov     AL,CL
[992] 1162:             add     AL,[DI+1]
[NA2] 1163:             cmp     AL,[DI]
[8H2] 1164:             ja      ExtConStr
[ZXØ] 1165:             pushf
[MF1] 1166:             push    SI
[PE1] 1167:             push    DI
[PF1] 1168:             push    ES
[MF1] 1169:             push    DS
[GD1] 117Ø:             pop     ES
[AD2] 1171:             mov     AL,[DI+1]
[LC2] 1172:             add     [DI+1],CL
[6T1] 1173:             xor     CH,CH
[1H1] 1174:             add     SI,2
[ØT1] 1175:             xor     AH,AH
[2P1] 1176:             add     DI,AX
[3G1] 1177:             add     DI,2
[ULØ] 1178:             cld
[9Y1] 1179:             rep     movsb
[GD1] 118Ø:             pop     ES
[GC1] 1181:             pop     DI
[ED1] 1182:             pop     SI
[LSØ] 1183:             popf
[RF2] 1184: ExtConStr:  ret
[XL2] 1185: ConcatStr   endp
[ØØØ] 1186:
[UC3] 1187: PosStr      proc    near
[ED2] 1188:             mov     CL,[DI+1]
[ØE2] 1189:             mov     CH,[SI+1]
[9L1] 119Ø:             or      CL,CL
[FM2] 1191:             jz      ExtPosStr
[UR1] 1192:             cmp     CL,CH
[4K2] 1193:             ja      ExtPosStr
[ZXØ] 1194:             pushf
[ULØ] 1195:             cld
[MF1] 1196:             push    SI
[PE1] 1197:             push    DI
[PF1] 1198:             push    ES
[MF1] 1199:             push    DS
[GD1] 12ØØ:             pop     ES
[1H1] 12Ø1:             add     SI,2
[3G1] 12Ø2:             add     DI,2
[PR1] 12Ø3:             sub     CH,CL
[6A1] 12Ø4:             inc     CH
[6M1] 12Ø5:             mov     AL,1
[G43] 12Ø6: PosLoop:    push    CX
[MF1] 12Ø7:             push    SI
[PE1] 12Ø8:             push    DI
[6T1] 12Ø9:             xor     CH,CH
[HY1] 121Ø:             repz    cmpsb
[UD2] 1211:             jz      PosFound
[GC1] 1212:             pop     DI
[ED1] 1213:             pop     SI
[VD1] 1214:             pop     CX
[EA1] 1215:             inc     AL
[9B1] 1216:             inc     SI
[191] 1217:             dec     CH
[PH2] 1218:             jnz     PosLoop
[GD1] 1219:             pop     ES
[GC1] 122Ø:             pop     DI
[ED1] 1221:             pop     SI
[LSØ] 1222:             popf
[MM3] 1223: ExtPosStr:  xor     AL,AL
[3PØ] 1224:             ret
[ØØ3] 1225: PosFound:   pop     DI
[ED1] 1226:             pop     SI
[VD1] 1227:             pop     CX
[GD1] 1228:             pop     ES
[GC1] 1229:             pop     DI
```

Listing 1. »system.asm« stellt Ihnen 40 nützliche Routinen zur Verfügung

```
[ED1] 123Ø:             pop     SI
[LSØ] 1231:             popf
[3PØ] 1232:             ret
[RB2] 1233: PosStr      endp
[ØØØ] 1234:
[E83] 1235: UpStr       proc    near
[CE2] 1236:             mov     CL,[SI+1]
[9L1] 1237:             or      CL,CL
[LK2] 1238:             jnz     OkUpStr
[3PØ] 1239:             ret
[YD3] 124Ø: OkUpStr:    xor     CH,CH
[ZXØ] 1241:             pushf
[MF1] 1242:             push    SI
[ULØ] 1243:             cld
[1H1] 1244:             add     SI,2
[LG2] 1245: UpLoop:     lodsb
[DT1] 1246:             cmp     AL,"a"
[9N1] 1247:             jb      ChkOk
[ZU1] 1248:             cmp     AL,"z"
[6N1] 1249:             ja      ChkOk
[W42] 125Ø:             and     AL,ØDFh
[1K2] 1251:             mov     [SI-1],AL
[SH3] 1252: ChkOk:      loop    UpLoop
[ED1] 1253:             pop     SI
[LSØ] 1254:             popf
[3PØ] 1255:             ret
[B72] 1256: UpStr       endp
[ØØØ] 1257:
[DE3] 1258: IgnChar     proc    near
[CE2] 1259:             mov     CL,[SI+1]
[9L1] 126Ø:             or      CL,CL
[XG2] 1261:             jz      ExtIgnChr
[MF1] 1262:             push    SI
[6T1] 1263:             xor     CH,CH
[1H1] 1264:             add     SI,2
[ØT1] 1265:             xor     AH,AH
[ZV3] 1266: IgnLoop:    cmp     [SI],AL
[AY1] 1267:             jnz     IgnOk
[2A1] 1268:             inc     AH
[9B1] 1269:             inc     SI
[7G2] 127Ø:             loop    IgnLoop
[NL2] 1271: IgnOk:      pop     SI
[FK1] 1272:             or      AH,AH
[XG2] 1273:             jz      ExtIgnChr
[6T1] 1274:             mov     AL,AH
[UL1] 1275:             mov     AH,1
[2N2] 1276:             jmp     DeleteStr
[1D2] 1277: ExtIgnChr:  ret
[AD2] 1278: IgnChar     endp
[ØØØ] 1279:
[3G3] 128Ø: CutChar     proc    near
[CE2] 1281:             mov     CL,[SI+1]
[9L1] 1282:             or      CL,CL
[5K2] 1283:             jz      ExtCutChr
[MF1] 1284:             push    SI
[6T1] 1285:             xor     CH,CH
[2R1] 1286:             add     SI,CX
[9B1] 1287:             inc     SI
[ØT1] 1288:             xor     AH,AH
[PT1] 1289:             mov     DL,CL
[NX3] 129Ø: CutLoop:    cmp     [SI],AL
[ØØ2] 1291:             jnz     CutOk
[2A1] 1292:             inc     AH
[4A1] 1293:             dec     SI
[F91] 1294:             dec     DL
[XH2] 1295:             loop    CutLoop
[KL2] 1296: CutOk:      inc     DL
[ED1] 1297:             pop     SI
[FK1] 1298:             or      AH,AH
[5K2] 1299:             jz      ExtCutChr
[6T1] 13ØØ:             mov     AL,AH
[5T1] 13Ø1:             mov     AH,DL
[2N2] 13Ø2:             jmp     DeleteStr
[9E2] 13Ø3: ExtCutChr:  ret
[ØF2] 13Ø4: CutChar     endp
[ØØØ] 13Ø5:
[8L3] 13Ø6: ParToStr    proc    near
[4Ø4] 13Ø7:             mov     byte ptr [SI+1],Ø
[BU2] 13Ø8:             mov     CL,ES:[DI]
[9L1] 13Ø9:             or      CL,CL
[GL2] 131Ø:             jz      ExtParStr
[RB2] 1311:             cmp     CL,[SI]
[5H2] 1312:             ja      ExtParStr
[ZXØ] 1313:             pushf
[MF1] 1314:             push    SI
[PE1] 1315:             push    DI
[MF1] 1316:             push    DS
[PF1] 1317:             push    ES
[MF1] 1318:             push    DS
[PF1] 1319:             push    ES
[ED1] 132Ø:             pop     DS
[GD1] 1321:             pop     ES
[BA1] 1322:             inc     DI
[TZ2] 1323:             mov     ES:[SI+1],CL
[1H1] 1324:             add     SI,2
[7T1] 1325:             xchg    SI,DI
[ULØ] 1326:             cld
[6T1] 1327:             xor     CH,CH
[9Y1] 1328:             rep     movsb
[GD1] 1329:             pop     ES
```

```
[ED1] 133Ø:             pop     DS
[GC1] 1331:             pop     DI
[ED1] 1332:             pop     SI
[LSØ] 1333:             popf
[MF2] 1334: ExtParStr:  ret
[5K2] 1335: ParToStr    endp
[ØØØ] 1336:
[FØ3] 1337: DAdd        proc    near
[AR1] 1338:             add     AX,BX
[DR1] 1339:             adc     DX,CX
[3PØ] 134Ø:             ret
[CZ1] 1341: DAdd        endp
[ØØØ] 1342:
[743] 1343: DSub        proc    near
[TT1] 1344:             sub     AX,BX
[7S1] 1345:             sbb     DX,CX
[3PØ] 1346:             ret
[432] 1347: DSub        endp
[ØØØ] 1348:
[Z33] 1349: DMul        proc    near
[MF1] 135Ø:             push    SI
[PE1] 1351:             push    DI
[3G1] 1352:             push    DX
[XF1] 1353:             push    AX
[ND1] 1354:             mul     BX
[6U1] 1355:             mov     DI,DX
[1V1] 1356:             mov     SI,AX
[RD1] 1357:             pop     AX
[RD1] 1358:             mul     CX
[2P1] 1359:             add     DI,AX
[RD1] 136Ø:             pop     AX
[ND1] 1361:             mul     BX
[2P1] 1362:             add     DI,AX
[NU1] 1363:             mov     DX,DI
[YU1] 1364:             mov     AX,SI
[GC1] 1365:             pop     DI
[ED1] 1366:             pop     SI
[3PØ] 1367:             ret
[W22] 1368: DMul        endp
[ØØØ] 1369:
[Y13] 137Ø: DDiv        proc    near
[ZF1] 1371:             push    BX
[3N1] 1372:             or      BX,CX
[TD1] 1373:             pop     BX
[CH2] 1374:             jnz     StartDiv
[ENØ] 1375:             stc
[3PØ] 1376:             ret
[F33] 1377: StartDiv:   push    BP
[MF1] 1378:             push    SI
[PE1] 1379:             push    DI
[2V1] 138Ø:             mov     SI,BX
[5U1] 1381:             mov     DI,CX
[7Z1] 1382:             mov     CX,2Øh
[LV1] 1383:             xor     BX,BX
[BU1] 1384:             xor     BP,BP
[TL1] 1385:             shl     AX,1
[EL1] 1386:             rcl     DX,1
[273] 1387: DivLoop:    rcl     BX,1
[KK1] 1388:             rcl     BP,1
[ZF1] 1389:             push    BX
[7F1] 139Ø:             push    BP
[ET1] 1391:             sub     BX,SI
[EP1] 1392:             sbb     BP,DI
[AW1] 1393:             jnc     DivOk
[1D1] 1394:             pop     BP
[TD1] 1395:             pop     BX
[PLØ] 1396:             clc
[YG2] 1397:             jmp     ContDivi
[AX2] 1398: DivOk:      add     SP,4
[ENØ] 1399:             stc
[C83] 14ØØ: ContDivi:   rcl     AX,1
[EL1] 14Ø1:             rcl     DX,1
[3H2] 14Ø2:             loop    DivLoop
[ZU1] 14Ø3:             mov     CX,BP
[GC1] 14Ø4:             pop     DI
[ED1] 14Ø5:             pop     SI
[1D1] 14Ø6:             pop     BP
[PLØ] 14Ø7:             clc
[3PØ] 14Ø8:             ret
[VØ2] 14Ø9: DDiv        endp
[ØØØ] 141Ø:
[X13] 1411: DNeg        proc    near
[3E1] 1412:             not     AX
[9E1] 1413:             not     DX
[9H1] 1414:             add     AX,1
[AH1] 1415:             adc     DX,Ø
[3PØ] 1416:             ret
[UØ2] 1417: DNeg        endp
[ØØØ] 1418:
[CM3] 1419: NumToStr    proc    near
[RK1] 142Ø:             cmp     BL,2
[592] 1421:             jb      BasisErr
[EW1] 1422:             cmp     BL,1Øh
[9B2] 1423:             jbe     BasisOk
[382] 1424: BasisErr:   stc
[3PØ] 1425:             ret
[T13] 1426: BasisOk:    push    AX
[ZF1] 1427:             push    BX
[7F1] 1428:             push    BP
[3G1] 1429:             push    DX
```

```
[MF1] 143Ø:             push    SI
[BU1] 1431:             xor     BP,BP
[TC2] 1432:             mov     Basis,BL
[PV1] 1433:             xor     CX,CX
[3T1] 1434:             xor     BH,BH
[P34] 1435: NTSLoop:    mov     BL,Basis
[LP1] 1436:             call    DDiv
[YN1] 1437:             add     BL,"Ø"
[ZS1] 1438:             cmp     BL,"9"
[U12] 1439:             jbe     ZiffOk
[DG1] 144Ø:             add     BL,7
[TY2] 1441: ZiffOk:     push    BX
[WA1] 1442:             inc     BP
[ZM1] 1443:             or      AX,AX
[1K2] 1444:             jnz     NTSLoop
[8N1] 1445:             or      DX,DX
[1K2] 1446:             jnz     NTSLoop
[ZU1] 1447:             mov     CX,BP
[RB2] 1448:             cmp     CL,[SI]
[XY1] 1449:             ja      NoZahl
[1K2] 145Ø:             mov     [SI+1],CL
[1H1] 1451:             add     SI,2
[EU2] 1452: GtLoop:     pop     AX
[UD2] 1453:             mov     [SI],AL
[9B1] 1454:             inc     SI
[872] 1455:             loop    GtLoop
[ED1] 1456:             pop     SI
[XD1] 1457:             pop     DX
[1D1] 1458:             pop     BP
[TD1] 1459:             pop     BX
[RD1] 146Ø:             pop     AX
[PLØ] 1461:             clc
[3PØ] 1462:             ret
[XU2] 1463: NoZahl:     pop     AX
[R72] 1464:             loop    NoZahl
[ED1] 1465:             pop     SI
[4Ø4] 1466:             mov     byte ptr [SI+1],Ø
[XD1] 1467:             pop     DX
[1D1] 1468:             pop     BP
[TD1] 1469:             pop     BX
[RD1] 147Ø:             pop     AX
[ENØ] 1471:             stc
[3PØ] 1472:             ret
[9L2] 1473: NumToStr    endp
[ØØØ] 1474:
[RM3] 1475: StrToNum    proc    near
[FG2] 1476:             mov     NegFlag,Ø
[GV1] 1477:             xor     AX,AX
[TV1] 1478:             xor     DX,DX
[CE2] 1479:             mov     CL,[SI+1]
[9L1] 148Ø:             or      CL,CL
[M82] 1481:             jnz     ZStrOk
[3PØ] 1482:             ret
[V13] 1483: ZStrOk:     push    BX
[1G1] 1484:             push    CX
[RT1] 1485:             mov     DI,SI
[6T1] 1486:             xor     CH,CH
[4P1] 1487:             add     DI,CX
[BA1] 1488:             inc     DI
[9C2] 1489:             mov     BL,[DI]
[Y42] 149Ø:             and     BL,ØDFh
[XT1] 1491:             cmp     BL,"H"
[TØ2] 1492:             jnz     NextZ
[MX1] 1493:             mov     BH,1Øh
[D91] 1494:             dec     CL
[ZK1] 1495:             cmp     CL,8
[VM1] 1496:             jbe     ZOk
[483] 1497:             jmp     short ZError
[U13] 1498: NextZ:      cmp     BL,"B"
[C52] 1499:             jnz     NextZ1
[XL1] 15ØØ:             mov     BH,2
[LW1] 15Ø1:             cmp     CL,21h
[8Z1] 15Ø2:             ja      ZError
[D91] 15Ø3:             dec     CL
[WL2] 15Ø4:             jmp     Short ZOk
[AV2] 15Ø5: ZError:     pop     CX
[TD1] 15Ø6:             pop     BX
[ENØ] 15Ø7:             stc
[3PØ] 15Ø8:             ret
[293] 15Ø9: NextZ1:     mov     BH,ØAh
[LX1] 151Ø:             cmp     CL,ØBh
[8Z1] 1511:             ja      ZError
[864] 1512:             cmp     byte ptr [SI+2],"-"
[4U1] 1513:             jnz     ZOk1
[D91] 1514:             dec     CL
[1Z1] 1515:             mov     BL,ØAh
[RT1] 1516:             mov     DI,SI
[4G1] 1517:             add     DI,3
[WM2] 1518:             mov     NegFlag,-1
[UB3] 1519:             jmp     short STNLoop
[GØ3] 152Ø: Zok1:       cmp     CL,ØBh
[L12] 1521:             jz      ZError
[2Y2] 1522: ZOk:        mov     BL,BH
[RT1] 1523:             mov     DI,SI
[3G1] 1524:             add     DI,2
[G43] 1525: STNLoop:    push    BX
[1G1] 1526:             push    CX
[PV1] 1527:             xor     CX,CX
[3T1] 1528:             xor     BH,BH
[MS1] 1529:             call    DMul
[VD1] 153Ø:             pop     CX
```

```
[TD1] 1531:             pop     BX
[XB2] 1532:             mov     BH,[DI]
[3T1] 1533:             cmp     BH,"a"
[7K1] 1534:             jb      ZfOk
[K42] 1535:             and     BH,ØDFh
[PV2] 1536: ZfOk:       sub     BH,"Ø"
[KK1] 1537:             cmp     BH,9
[3M1] 1538:             jbe     NOk
[GK1] 1539:             sub     BH,7
[MM2] 154Ø: NOk:        push    BX
[9T1] 1541:             mov     BL,BH
[3T1] 1542:             xor     BH,BH
[AR1] 1543:             add     AX,BX
[AH1] 1544:             adc     DX,Ø
[TD1] 1545:             pop     BX
[BA1] 1546:             inc     DI
[2K2] 1547:             loop    STNLoop
[XE2] 1548:             cmp     NegFlag,Ø
[XD2] 1549:             jz      ZPositiv
[KP1] 155Ø:             call    DNeg
[Ø13] 1551: ZPositiv:   pop     CX
[TD1] 1552:             pop     BX
[PLØ] 1553:             clc
[3PØ] 1554:             ret
[ML2] 1555: StrToNum    endp
[ØØØ] 1556:
[ØT2] 1557: ; Hilfsroutinen
[ØØØ] 1558:
[SM3] 1559: BerechneXY  proc    near
[4A1] 156Ø:             push    AX
[6A1] 1561:             push    CX
[7A1] 1562:             push    DX
[NP1] 1563:             xor     DH,DH
[WR1] 1564:             mov     AX,DX
[UK1] 1565:             mov     CL,7
[VN1] 1566:             shl     AX,CL
[NP1] 1567:             mov     DI,AX
[ØL1] 1568:             add     DI,DX
[ØL1] 1569:             add     DI,DX
[ØL1] 157Ø:             add     DI,DX
[ØL1] 1571:             add     DI,DX
[ØL1] 1572:             add     DI,DX
[ØL1] 1573:             add     DI,DX
[ØL1] 1574:             add     DI,DX
[ØL1] 1575:             add     DI,DX
[ØL1] 1576:             add     DI,DX
[ØL1] 1577:             add     DI,DX
[ØL1] 1578:             add     DI,DX
[ØL1] 1579:             add     DI,DX
[ØL1] 158Ø:             add     DI,DX
[ØL1] 1581:             add     DI,DX
[ØL1] 1582:             add     DI,DX
[ØL1] 1583:             add     DI,DX
[ØL1] 1584:             add     DI,DX
[ØL1] 1585:             add     DI,DX
[ØL1] 1586:             add     DI,DX
[ØL1] 1587:             add     DI,DX
[ØL1] 1588:             add     DI,DX
[ØL1] 1589:             add     DI,DX
[ØL1] 159Ø:             add     DI,DX
[ØL1] 1591:             add     DI,DX
[ØL1] 1592:             add     DI,DX
[ØL1] 1593:             add     DI,DX
[ØL1] 1594:             add     DI,DX
[ØL1] 1595:             add     DI,DX
[ØL1] 1596:             add     DI,DX
[ØL1] 1597:             add     DI,DX
[ØL1] 1598:             add     DI,DX
[ØL1] 1599:             add     DI,DX
[181] 16ØØ:             pop     DX
[MP1] 16Ø1:             mov     AL,DH
[6P1] 16Ø2:             xor     AH,AH
[7H1] 16Ø3:             shl     AX,1
[MK1] 16Ø4:             add     DI,AX
[Ø81] 16Ø5:             pop     CX
[Y71] 16Ø6:             pop     AX
[NG2] 16Ø7:             mov     ES,ScrSeg
[3PØ] 16Ø8:             ret
[3R2] 16Ø9: BerechneXY  endp
[ØØØ] 161Ø:
[XM3] 1611: SelectWin   proc    near
[ZXØ] 1612:             pushf
[PF1] 1613:             push    ES
[MF1] 1614:             push    DS
[GD1] 1615:             pop     ES
[ULØ] 1616:             cld
[SS2] 1617:             mov     AL,WSelect
[ØT1] 1618:             xor     AH,AH
[CM1] 1619:             mov     CL,3
[3T1] 162Ø:             shl     AX,CL
[4W3] 1621:             add     AX,offset Windows
[1V1] 1622:             mov     SI,AX
[TØ4] 1623:             mov     DI,offset WinYPos
[HN1] 1624:             mov     CX,4
[LZ1] 1625:             rep     movsw
[GD1] 1626:             pop     ES
[LSØ] 1627:             popf
[UL2] 1628: SelectWin   endp
```

Listing 1. »system.asm« stellt Ihnen 40 nützliche Routinen zur Verfügung

```
[ØØØ] 1629:
[RP3] 163Ø: SetCursor   proc    near
[3H2] 1631:             cmp     CurFlag,Ø
[XL2] 1632:             jz      CursorVis
[VL1] 1633:             mov     AH,2
[3T1] 1634:             xor     BH,BH
[U52] 1635:             mov     DX,19ØØh
[YH1] 1636:             int     1Øh
[3PØ] 1637:             ret
[VL5] 1638: CursorVis:  mov     AX,word ptr ActY
[GV1] 1639:             mov     DX,AX
[2A1] 164Ø:             inc     AH
[4B2] 1641:             cmp     AH,Width
[RM1] 1642:             jbe     XOk
[9T1] 1643:             xor     DH,DH
[MA1] 1644:             inc     DL
[EA1] 1645:             inc     AL
[AD2] 1646: XOk:        inc     AL
[KG2] 1647:             mov     CL,Height
[YR1] 1648:             cmp     AL,CL
[TM1] 1649:             jbe     YOk
[PT1] 165Ø:             mov     DL,CL
[F91] 1651:             dec     DL
[YR1] 1652:             sub     AL,CL
[991] 1653:             dec     AL
[DK1] 1654:             jz      YOk
[KT1] 1655:             mov     CL,AL
[6T1] 1656:             xor     CH,CH
[3G1] 1657:             push    DX
[193] 1658: ScrCrLoop:  push    CX
[BH2] 1659:             call    ScrollUp
[VD1] 166Ø:             pop     CX
[YR2] 1661:             loop    ScrCrloop
[XD1] 1662:             pop     DX
[955] 1663: YOk:        mov     word ptr ActY,DX
[GG4] 1664:             add     DX,word ptr WinYPos
[CS1] 1665:             xchg    DH,DL
[3T1] 1666:             xor     BH,BH
[VL1] 1667:             mov     AH,2
[YH1] 1668:             int     1Øh
[3PØ] 1669:             ret
[MN2] 167Ø: SetCursor   endp
[ØØØ] 1671:
[ME3] 1672: SaveBack    proc    near
[ZXØ] 1673:             pushf
[MF1] 1674:             push    SI
[MF1] 1675:             push    DS
[PF1] 1676:             push    ES
[ULØ] 1677:             cld
[EY3] 1678:             mov     DX,word ptr [SI]
[CS1] 1679:             xchg    DH,DL
[VV2] 168Ø:             call    BerechneXY
[N84] 1681:             mov     CX,word ptr [SI+2]
[9S1] 1682:             xchg    CH,CL
[M84] 1683:             mov     SI,word ptr [SI+8]
[7T1] 1684:             xchg    SI,DI
[ULØ] 1685:             cld
[MF1] 1686:             push    DS
[PF1] 1687:             push    ES
[ED1] 1688:             pop     DS
[GD1] 1689:             pop     ES
[7Z2] 169Ø: SBLoop:     push    CX
[MF1] 1691:             push    SI
[6T1] 1692:             xor     CH,CH
[LZ1] 1693:             rep     movsw
[ED1] 1694:             pop     SI
[H22] 1695:             add     SI,ØAØh
[VD1] 1696:             pop     CX
[191] 1697:             dec     CH
[Z52] 1698:             jnz     SBLoop
[GD1] 1699:             pop     ES
[ED1] 17ØØ:             pop     DS
[ED1] 17Ø1:             pop     SI
[LSØ] 17Ø2:             popf
[3PØ] 17Ø3:             ret
[HD2] 17Ø4: SaveBack    endp
[ØØØ] 17Ø5:
[6L3] 17Ø6: ChkInsert   proc    near
[ZXØ] 17Ø7:             pushf
[XF1] 17Ø8:             push    AX
[ZF1] 17Ø9:             push    BX
[1G1] 171Ø:             push    CX
[3G1] 1711:             push    DX
[MF1] 1712:             push    SI
[PE1] 1713:             push    DI
[UC2] 1714:             cmp     Insert,Ø
[GH2] 1715:             jz      ExtChkIns
[8K2] 1716:             mov     AL,BufPos
[PC2] 1717:             cmp     AL,[SI+1]
[GH2] 1718:             jz      ExtChkIns
[5D2] 1719:             mov     AL,[SI]
[ØT1] 172Ø:             xor     AH,AH
[ØR1] 1721:             add     SI,AX
[9B1] 1722:             inc     SI
[RT1] 1723:             mov     DI,SI
[BA1] 1724:             inc     DI
[EV1] 1725:             mov     CX,AX
[UG2] 1726:             sub     CL,BufPos
[KNØ] 1727:             std
[9Y1] 1728:             rep     movsb
[G23] 1729: ExtChkIns:  pop     DI
[ED1] 173Ø:             pop     SI
[XD1] 1731:             pop     DX
[VD1] 1732:             pop     CX
[TD1] 1733:             pop     BX
[RD1] 1734:             pop     AX
[LSØ] 1735:             popf
[3PØ] 1736:             ret
[3K2] 1737: ChkInsert   endp
[ØØØ] 1738:
[4B3] 1739: GetPos      proc    near
[1G1] 174Ø:             push    CX
[8K2] 1741:             mov     AL,BufPos
[ØT1] 1742:             xor     AH,AH
[6D2] 1743:             mov     CL,Width
[5C1] 1744:             div     CL
[ZG2] 1745:             mov     DX,PosXY
[FR1] 1746:             add     DX,AX
[VD1] 1747:             pop     CX
[3PØ] 1748:             ret
[1A2] 1749: GetPos      endp
[ØØØ] 175Ø:
[CP3] 1751: FreeBuffer  proc    near
[CE2] 1752:             mov     CL,[SI+1]
[9L1] 1753:             or      CL,CL
[HG2] 1754:             jnz     DelAllOk
[3PØ] 1755:             ret
[PB3] 1756: DelAllOk:   xor     CH,CH
[RT1] 1757:             mov     DI,SI
[3G1] 1758:             add     DI,2
[ZX1] 1759:             mov     AL,2Øh
[PF1] 176Ø:             push    ES
[MF1] 1761:             push    DS
[GD1] 1762:             pop     ES
[ULØ] 1763:             cld
[8Y1] 1764:             rep     stosb
[GD1] 1765:             pop     ES
[ZG2] 1766:             mov     DX,PosXY
[LV3] 1767:             xchg    DX,word ptr ActY
[3G1] 1768:             push    DX
[NX1] 1769:             call    Print
[V93] 177Ø:             pop     word ptr ActY
[8U1] 1771:             mov     BP,SI
[MG1] 1772:             add     BP,2
[1D2] 1773:             mov     BufPos,Ø
[4Ø4] 1774:             mov     byte ptr [SI+1],Ø
[T42] 1775:             call    GetPos
[TT3] 1776:             mov     word ptr ActY,DX
[MF1] 1777:             push    SI
[PE1] 1778:             push    DI
[7F1] 1779:             push    BP
[CR2] 178Ø:             call    SetCursor
[1D1] 1781:             pop     BP
[GC1] 1782:             pop     DI
[ED1] 1783:             pop     SI
[3PØ] 1784:             ret
[9N2] 1785: FreeBuffer  endp
[ØØØ] 1786:
[112] 1787: Code        ends
[ØØØ] 1788:
[4MØ] 1789:             end
```

Listing 1. »system.asm« stellt Ihnen 40 nützliche Routinen zur Verfügung

```
[4G5]    1: 8Ø ØC ØØ ØA 73 79 73 74 65 6D 2E 41 53 4D B6 96
[695]    2: ØC ØØ ØØ Ø4 44 41 54 41 Ø4 43 4F 44 45 21 98 Ø7
[195]    3: ØØ 48 61 Ø1 Ø2 Ø1 Ø1 B3 98 Ø7 ØØ 28 A8 ØC Ø3 Ø1
[G75]    4: Ø1 8Ø 9Ø ØD ØØ ØØ Ø1 Ø6 53 43 52 53 45 47 Ø1 ØØ
[595]    5: ØØ 94 9Ø ØD ØØ ØØ Ø2 Ø6 43 4C 52 53 43 52 68 ØØ
[Y75]    6: ØØ 2A 9Ø 21 ØØ ØØ Ø1 Ø6 42 55 46 5Ø 4F 53 54 Ø1
[7B5]    7: ØØ Ø6 49 4E 53 45 52 54 4F ØØ ØØ Ø6 53 54 52 42
[ØC5]    8: 55 46 5Ø ØØ ØØ CE 9Ø 17 ØØ ØØ Ø2 Ø6 47 4F 54 4F
[GC5]    9: 58 59 A2 Ø3 ØØ Ø6 5Ø 4F 53 53 54 52 C6 Ø8 ØØ Ø3
[E85]   1Ø: 9Ø 23 ØØ ØØ Ø1 Ø7 44 45 4C 46 4C 41 47 5D Ø1 ØØ
[ZC5]   11: Ø7 4E 45 47 46 4C 41 47 5F Ø1 ØØ Ø6 53 59 53 53
[985]   12: 54 52 59 Ø1 ØØ 45 9Ø 19 ØØ ØØ Ø2 Ø7 49 47 4E 43
[EF5]   13: 48 41 52 2C Ø9 ØØ Ø7 44 45 4C 4C 49 4E 45 BA ØØ
[WD5]   14: ØØ 5F 9Ø 19 ØØ ØØ Ø1 Ø7 4B 45 59 46 4C 41 47 4C
[1A5]   15: ØØ ØØ Ø7 43 55 52 46 4C 41 47 ØA ØØ ØØ EB 9Ø 19
[885]   16: ØØ ØØ Ø2 Ø7 52 45 41 44 4B 45 59 57 Ø4 ØØ Ø7 43
[N95]   17: 55 54 43 48 41 52 51 Ø9 ØØ 83 9Ø ØE ØØ ØØ Ø1 Ø7
[ØB5]   18: 53 43 52 4D 4F 44 45 ØØ ØØ ØØ 4D 9Ø ØE ØØ ØØ Ø2
[G95]   19: Ø7 49 4E 53 4C 49 4E 45 11 Ø1 ØØ 35 9Ø 19 ØØ ØØ
[V95]   2Ø: Ø1 Ø7 4F 55 54 46 4C 41 47 5C Ø1 ØØ Ø7 57 53 45
[KD5]   21: 4C 45 43 54 ØB ØØ ØØ B7 9Ø ØE ØØ ØØ Ø2 Ø7 49 4E
[A75]   22: 49 54 53 43 52 2F ØØ ØØ ØE 9Ø 24 ØØ ØØ Ø1 Ø7 41
[UD5]   23: 4E 5A 4B 45 59 53 55 Ø1 ØØ Ø7 57 49 4E 41 54 54
[DB5]   24: 52 Ø9 ØØ ØØ Ø7 57 49 4E 44 4F 57 53 ØC ØØ ØØ 52
[A95]   25: 9Ø ØE ØØ ØØ Ø2 Ø7 43 4F 5Ø 59 53 54 52 48 Ø8 ØØ
[P95]   26: D5 9Ø 19 ØØ ØØ Ø1 Ø7 57 49 4E 58 5Ø 4F 53 Ø4 ØØ
[YC5]   27: ØØ Ø7 57 49 4E 59 5Ø 4F 53 Ø3 ØØ ØØ DØ 9Ø A1 ØØ
[Ø95]   28: ØØ Ø2 Ø8 44 45 4C 46 52 41 4D 45 47 Ø2 ØØ Ø8 47
[UG5]   29: 45 54 43 4F 4C 4F 52 6B Ø1 ØØ Ø8 53 43 52 4F 4C
[ØM5]   3Ø: 4C 4C 46 FD Ø2 ØØ Ø8 53 43 52 4F 4C 4C 44 4E B9
[485]   31: Ø2 ØØ Ø8 5Ø 55 54 46 52 41 4D 45 FØ Ø1 ØØ Ø8 53
[VD5]   32: 43 52 4F 4C 4C 52 47 53 Ø3 ØØ Ø8 53 45 54 43 4F
[8F5]   33: 4C 4F 52 67 Ø1 ØØ Ø8 53 43 52 4F 4C 4C 55 5Ø 77
[2B5]   34: Ø2 ØØ Ø8 5Ø 41 52 54 4F 53 54 52 7C Ø9 ØØ Ø8 53
[PL5]   35: 54 52 54 4F 4E 55 4D 74 ØA ØØ Ø8 4E 55 4D 54 4F
[595]   36: 53 54 52 16 ØA ØØ Ø9 44 45 54 45 43 54 53 43 52
[2C5]   37: ØØ ØØ ØØ Ø9 44 45 46 57 49 4E 44 4F 57 6F Ø1 ØØ
[T85]   38: 4E 9Ø 1Ø ØØ ØØ Ø1 Ø9 53 43 52 55 5Ø 46 4C 41 47
[1A5]   39: 6Ø Ø1 ØØ 4E 9Ø 64 ØØ ØØ Ø2 Ø9 44 45 4C 45 54 45
[6B5]   4Ø: 53 54 52 95 Ø7 ØØ Ø9 43 4F 4E 43 41 54 53 54 52
```

```
[1E5]   41: 96 Ø8 ØØ Ø9 41 53 4B 57 49 4E 44 4F 57 CE Ø1 ØØ
[FH5]   42: Ø9 53 45 4C 57 49 4E 44 4F 57 E6 Ø1 ØØ Ø9 49 4E
[AD5]   43: 53 45 52 54 53 54 52 EB Ø7 ØØ Ø4 44 41 44 44 A9
[LF5]   44: Ø9 ØØ Ø4 44 4E 45 47 ØB ØA ØØ Ø4 44 44 49 56 CE
[Z95]   45: Ø9 ØØ Ø4 44 53 55 42 AE Ø9 ØØ 78 9Ø 13 ØØ ØØ Ø1
[D65]   46: Ø4 41 43 54 58 Ø8 ØØ ØØ Ø4 41 43 54 59 Ø7 ØØ ØØ
[CA5]   47: E4 9Ø 19 ØØ ØØ Ø2 Ø4 44 4D 55 4C B3 Ø9 ØØ ØA 43
[GD5]   48: 4F 4E 56 53 54 52 49 4E 47 29 Ø4 ØØ 1F 9Ø 15 ØØ
[C85]   49: ØØ Ø1 Ø5 42 41 53 49 53 5E Ø1 ØØ Ø5 57 49 44 54
[895]   5Ø: 48 Ø6 ØØ ØØ F9 9Ø 3Ø ØØ ØØ Ø2 Ø5 46 4C 55 53 48
[1B5]   51: A3 Ø4 ØØ Ø5 5Ø 52 49 4E 54 BE Ø3 ØØ Ø5 49 4E 5Ø
[LB5]   52: 55 54 B4 Ø4 ØØ Ø5 47 45 54 58 59 A9 Ø3 ØØ Ø5 55
[395]   53: 5Ø 53 54 52 Ø9 Ø9 ØØ 79 9Ø 2Ø ØØ ØØ Ø1 Ø5 5Ø 4F
[Z75]   54: 53 58 59 57 Ø1 ØØ Ø6 42 55 46 41 44 52 4D ØØ ØØ
[B95]   55: Ø6 48 45 49 47 48 54 Ø5 ØØ ØØ 84 9Ø ØD ØØ ØØ Ø2
[TA5]   56: Ø6 43 4C 52 45 4F 4C 97 ØØ ØØ Ø3 88 Ø4 ØØ ØØ A2
[225]   57: Ø1 D1 AØ 1Ø ØØ Ø1 ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[S55]   58: ØØ ØØ ØØ ØØ 4F AØ Ø8 ØØ Ø1 4C ØØ ØØ ØØ ØØ ØØ ØB
[815]   59: AØ 11 ØØ Ø1 54 Ø1 ØØ ØØ ØØ ØØ ØØ Ø1 Ø1 ØØ ØØ ØØ
[Z95]   6Ø: ØØ ØØ ØØ F7 AØ ØØ Ø4 Ø2 ØØ ØØ 51 Ø6 33 C9 8E C1
[NE5]   61: 26 8B ØE 63 Ø4 81 F9 D4 Ø3 Ø7 59 74 ØE C6 Ø6 ØØ
[CB5]   62: ØØ Ø7 C7 Ø6 ØØ ØØ ØØ BØ 32 CØ C3 C6 Ø6 ØØ ØØ Ø3
[ØA5]   63: C7 Ø6 ØØ ØØ ØØ B8 BØ Ø1 C3 AØ ØØ ØØ ØA CØ 75 Ø1
[HK5]   64: C3 32 E4 CD 1Ø BE ØØ ØØ BF ØØ ØØ C7 Ø4 ØØ ØØ C7
[G85]   65: 44 Ø2 19 5Ø C7 44 Ø4 ØØ ØØ C7 44 Ø6 Ø7 ØØ 9C Ø6
[1P5]   66: FC 1E Ø7 B9 1C ØØ F3 A5 Ø7 9D C6 Ø6 ØØ ØØ ØØ E9
[4E5]   67: ØØ ØØ 9C Ø6 8B 16 ØØ ØØ E8 ØØ ØØ 8A 26 ØØ ØØ FC
[HP5]   68: 8B ØE ØØ ØØ 86 E9 51 57 32 ED F3 AB 5F 59 81 C7
[1K5]   69: AØ ØØ FE CD 75 FØ Ø7 9D C7 Ø6 ØØ ØØ ØØ ØØ E9 ØØ
[DC5]   7Ø: ØØ 9C Ø6 8B 16 ØØ ØØ Ø3 16 ØØ ØØ E8 ØØ ØØ 8A ØE
[SG5]   71: ØØ ØØ 2A ØE ØØ ØØ 32 ED BØ 2Ø 8A 26 ØØ ØØ FC F3
[ML5]   72: AB Ø7 9D C3 9C Ø6 FC 8B 16 ØØ ØØ Ø2 16 ØØ ØØ E8
[SG5]   73: ØØ ØØ AØ ØØ ØØ FE CØ 3A Ø6 ØØ ØØ 74 2C 8B F7 81
[1P5]   74: C6 AØ ØØ 8B ØE ØØ ØØ 86 E9 FE CD 2A 2E ØØ ØØ 1E
[PL5]   75: Ø6 1F 51 56 57 32 ED F3 A5 5F 81 C7 AØ ØØ 5E 81
[EP5]   76: C6 AØ ØØ 59 FE CD 75 EA 1F 8A ØE ØØ ØØ 32 ED BØ
[5P5]   77: 2Ø 8A 26 ØØ ØØ FC F3 AB Ø7 9D C3 9C Ø6 FC 8B 16
[X85]   78: ØØ ØØ Ø2 16 ØØ ØØ E8 ØØ ØØ AØ ØØ ØØ FE CØ 3A Ø6
[FD5]   79: ØØ ØØ 74 D5 57 8B 16 ØØ ØØ Ø2 16 ØØ ØØ FE CA E8
[NN5]   8Ø: ØØ ØØ 8B F7 81 EE AØ ØØ 8B ØE ØØ ØØ 86 E9 FE CD
[ZG5]   81: 2A 2E ØØ ØØ 1E Ø6 1F 51 56 57 32 ED F3 A5 5F 81
[ZV5]   82: EF AØ ØØ 5E 81 EE AØ ØØ 59 FE CD 75 EA 1F 5F EB
[KF5]   83: 98 A2 ØØ ØØ C3 AØ ØØ ØØ C3 ØA E4 75 Ø2 F9 C3 ØA
[HU5]   84: ED 74 FA ØA C9 74 F6 8Ø FE 4F 77 F1 8Ø FA 18 77
[A36]   85: EC 8B FA Ø2 F5 Ø2 D1 FE CE FE CA 8Ø FE 4F 77 DD
[AS5]   86: 8Ø FA 18 77 D8 8B D7 8Ø E4 Ø7 5Ø 8A DC 32 FF D1
[VK5]   87: E3 D1 E3 D1 E3 81 C3 ØØ ØØ 58 89 17 89 4F Ø2 C7
[9A5]   88: 47 Ø4 ØØ ØØ 88 47 Ø6 C6 47 Ø7 ØØ 3A 26 ØØ ØØ 74
[ZR5]   89: Ø2 F8 C3 E8 ØØ ØØ F8 C3 8Ø E4 Ø7 8A DC 32 FF B1
[PG5]   9Ø: Ø3 D3 E3 81 C3 ØØ ØØ 8B 17 8B 4F Ø2 8A 47 Ø6 C3
[VB5]   91: 8Ø E4 Ø7 88 26 ØØ ØØ E9 ØØ ØØ 8Ø 7C Ø5 Ø1 74 Ø1
[XK5]   92: C3 E8 ØØ ØØ C6 44 Ø5 ØØ 8B 14 86 F2 9C Ø6 FC E8
[XH5]   93: ØØ ØØ 8B 4C Ø2 86 E9 8A 64 Ø4 56 8B 74 Ø6 51 57
[XL5]   94: 32 ED AC ØA CØ 74 19 3C Ø1 74 1A 3C Ø2 74 1B AB
[DP5]   95: E2 FØ 5F 81 C7 AØ ØØ 59 FE CD 75 E2 5E 81 C7 9D
[ZU5]   96: AC 8A EØ EB DD 83 C7 Ø2 EB E6 26 8Ø 65 Ø1 Ø7 EB
[4E5]   97: F4 8Ø 7C Ø5 ØØ 74 Ø1 C3 C6 44 Ø5 Ø1 8B 14 86 F2
[CP5]   98: 9C Ø6 FC E8 ØØ ØØ 8B 4C Ø2 86 E9 8B 74 Ø8 51 57
[TU5]   99: 32 ED F3 A5 5F 81 C7 AØ ØØ 59 FE CD 75 FØ Ø7 9D
[YD5]   1ØØ: C3 9C Ø6 8B 16 ØØ ØØ E8 ØØ ØØ 8Ø 3E ØØ ØØ Ø1 75
[PH5]   1Ø1: Ø3 E9 ØØ ØØ FE Ø6 ØØ ØØ 8B ØE ØØ ØØ 86 E9 FE CD
[AM5]   1Ø2: 8B F7 81 C6 AØ ØØ 1E Ø6 1F 51 56 57 32 ED F3 A5
[5V5]   1Ø3: 5F 81 C7 AØ ØØ 5E 81 C6 AØ ØØ 59 FE CD 75 EA 1F
[YE5]   1Ø4: E9 ØØ ØØ 9C Ø6 8B 16 ØØ ØØ Ø2 16 ØØ ØØ FE CA E8
[795]   1Ø5: ØØ ØØ 8Ø 3E ØØ ØØ Ø1 75 Ø3 E9 ØØ ØØ 8B ØE ØØ ØØ
[ØT5]   1Ø6: 86 E9 FE CD 8B F7 81 EE AØ ØØ 1E Ø6 1F 51 56 57
[PV5]   1Ø7: 32 ED F3 A5 5F 81 EF AØ ØØ 5E 81 EE AØ ØØ 59 FE
[CH5]   1Ø8: CD 75 EA 1F E9 ØØ ØØ 9C Ø6 8B 16 ØØ ØØ E8 ØØ ØØ
[2E5]   1Ø9: 8Ø 3E ØØ ØØ Ø1 75 15 8A ØE ØØ ØØ 32 ED BØ 2Ø 8A
[BR5]   11Ø: 26 ØØ ØØ AB 83 C7 7E E2 FA Ø7 9D C3 8B F7 83 C6
[KK5]   111: Ø2 8B ØE ØØ ØØ 86 E9 FE C9 BØ 2Ø 8A 26 ØØ ØØ 1E
[KM5]   112: Ø6 1F 51 56 57 32 ED F3 A5 AB 5F 81 C7 AØ ØØ 5E
[CR5]   113: 81 C6 AØ ØØ 59 FE CD 75 E9 1F Ø7 9D C3 9C Ø6 8B
[SD5]   114: 16 ØØ ØØ E8 ØØ ØØ 8Ø 3E ØØ ØØ Ø1 75 Ø2 EB A8 AØ
[SP5]   115: ØØ ØØ FE C8 32 E4 D1 EØ Ø3 F8 8B F7 83 EE Ø2 8B
[7H5]   116: ØE ØØ ØØ 86 E9 FE C9 BØ 2Ø 8A 26 ØØ ØØ 1E Ø6 1F
[5R5]   117: FD 51 56 57 32 ED F3 A5 AB 5F 81 C7 AØ ØØ 5E 81
[DM5]   118: C6 AØ ØØ 59 FE CD 75 E9 1F Ø7 9D C3 89 16 ØØ ØØ
[9A5]   119: E9 ØØ ØØ 8B 16 ØØ ØØ C3 C6 Ø6 ØØ ØØ ØØ E9 ØØ ØØ
[V95]   12Ø: C6 Ø6 ØØ ØØ Ø1 E9 ØØ ØØ 9C 56 Ø6 8B 16 ØØ ØØ 52
[ME5]   121: Ø3 16 ØØ ØØ E8 ØØ ØØ 5A 46 AC ØA CØ 74 43 8A C8
[VN5]   122: 32 ED 8A 26 ØØ ØØ AC 3C ØD 74 3A 3C ØA 74 3A AB
[EF5]   123: FE C6 3A 36 ØØ ØØ 72 2Ø 32 F6 FE C2 3A 16 ØØ ØØ
[VN5]   124: 72 ØD FE CA 5Ø 51 EØ 9C 78 Ø2 C4 15 Ø4 Ø1 Ø1 C4
[S75]   125: 1A ØØ Ø1 Ø1 Ø1 ØØ C4 23 Ø4 Ø1 Ø1 C4 28 ØØ Ø1 Ø1
[2E5]   126: Ø1 ØØ C4 3Ø Ø4 Ø1 Ø1 C4 3C ØØ Ø1 Ø1 ØC ØØ C4 3F
[S35]   127: ØØ Ø1 Ø1 14 ØØ C4 62 ØØ Ø1 Ø1 ØB ØØ 84 66 ØØ Ø2
[GC5]   128: Ø2 79 ØB C4 6C ØØ Ø1 Ø1 Ø3 ØØ 84 6F ØØ Ø2 Ø2 1C
[185]   129: ØB C4 73 ØØ Ø1 Ø1 Ø9 ØØ C4 78 ØØ Ø1 Ø1 Ø5 ØØ C4
[Z85]   13Ø: 9Ø ØØ Ø1 Ø1 Ø7 ØØ 84 95 ØØ Ø2 Ø2 96 ØB C4 9B ØØ
[U55]   131: Ø1 Ø1 Ø7 ØØ C4 9F ØØ Ø1 Ø1 Ø3 ØØ 84 A2 ØØ Ø2 Ø2
[KB5]   132: 1C ØB C4 A6 ØØ Ø1 Ø1 Ø6 ØØ C4 AA ØØ Ø1 Ø1 Ø8 ØØ
[LC5]   133: C4 B2 ØØ Ø1 Ø1 Ø9 ØØ C4 BF ØØ Ø1 Ø1 Ø3 ØØ C4 C3
[595]   134: ØØ Ø1 Ø1 Ø7 ØØ 84 C6 ØØ Ø2 Ø2 1C ØB C4 C9 ØØ Ø1
[1D5]   135: Ø1 Ø7 ØØ C4 CF ØØ Ø1 Ø1 Ø5 ØØ C4 DB ØØ Ø1 Ø1 Ø5
[N75]   136: ØØ C4 E3 ØØ Ø1 Ø1 Ø7 ØØ C5 Ø1 ØØ Ø1 Ø1 Ø6 ØØ C5
[865]   137: Ø9 ØØ Ø1 Ø1 Ø9 ØØ C5 16 ØØ Ø1 Ø1 Ø3 ØØ C5 1A ØØ
[H65]   138: Ø1 Ø1 Ø7 ØØ 85 1D ØØ Ø2 Ø2 1C ØB C5 2Ø ØØ Ø1 Ø1
[W55]   139: Ø7 ØØ C5 26 ØØ Ø1 Ø1 Ø5 ØØ C5 2D ØØ Ø1 Ø1 Ø3 ØØ
[N95]   14Ø: C5 31 ØØ Ø1 Ø1 Ø5 ØØ 85 36 ØØ Ø2 Ø2 1C ØB C5 4Ø
[745]   141: ØØ Ø1 Ø1 Ø5 ØØ C5 48 ØØ Ø1 Ø1 Ø7 ØØ C5 68 ØØ Ø1
[AE5]   142: Ø1 Ø9 ØØ C5 6C ØØ Ø1 Ø1 Ø9 ØØ C5 AD ØØ Ø1 Ø1 ØC
[X95]   143: ØØ C5 C3 ØØ Ø1 Ø1 ØB ØØ 85 CA ØØ Ø2 Ø2 79 ØB C5
[UG5]   144: DB ØØ Ø1 Ø1 ØC ØØ C5 EB ØØ Ø1 Ø1 ØB ØØ 85 EE ØØ
[3A5]   145: Ø2 Ø2 79 ØB 85 F8 ØØ Ø2 Ø2 EC ØB 86 Ø6 ØØ Ø2 Ø2
[SC5]   146: 1C ØB 86 5A ØØ Ø2 Ø2 1C ØB C6 7B ØØ Ø1 Ø1 Ø3 ØØ
[LA5]   147: 86 7E ØØ Ø2 Ø2 1C ØB C6 82 ØØ Ø1 Ø1 Ø5 ØØ 86 88
[S95]   148: ØØ Ø2 Ø2 FF ØØ C6 8C ØØ Ø1 Ø1 6Ø Ø1 C6 9Ø ØØ Ø1
[ND5]   149: Ø1 Ø5 ØØ 86 B7 ØØ Ø2 Ø2 FF ØØ C6 BD ØØ Ø1 Ø1 Ø3
[F95]   15Ø: ØØ C6 C1 ØØ Ø1 Ø1 Ø5 ØØ 86 C6 ØØ Ø2 Ø2 1C ØB C6
[2E5]   151: CA ØØ Ø1 Ø1 Ø5 ØØ 86 DØ ØØ Ø2 Ø2 FF ØØ C6 D4 ØØ
[L75]   152: Ø1 Ø1 Ø5 ØØ 86 FB ØØ Ø2 Ø2 FF ØØ C7 Ø1 ØØ Ø1 Ø1
```

```
[Z65]   153: Ø3 ØØ 87 Ø4 ØØ Ø2 Ø2 1C ØB C7 Ø8 ØØ Ø1 Ø1 Ø6 ØØ
[1B5]   154: C7 ØF ØØ Ø1 Ø1 Ø5 ØØ C7 17 ØØ Ø1 Ø1 Ø9 ØØ C7 29
[745]   155: ØØ Ø1 Ø1 Ø5 ØØ C7 33 ØØ Ø1 Ø1 Ø9 ØØ C7 57 ØØ Ø1
[MC5]   156: Ø1 Ø3 ØØ 87 5A ØØ Ø2 Ø2 1C ØB C7 5E ØØ Ø1 Ø1 Ø6
[Ø75]   157: ØØ C7 66 ØØ Ø1 Ø1 Ø6 ØØ C7 77 ØØ Ø1 Ø1 Ø5 ØØ C7
[995]   158: 81 ØØ Ø1 Ø1 Ø9 ØØ C7 A4 ØØ Ø1 Ø1 Ø7 ØØ 87 A7 ØØ
[ZA5]   159: Ø2 Ø2 96 ØB C7 AB ØØ Ø1 Ø1 Ø7 ØØ C7 BØ ØØ Ø1 Ø1
[PB5]   16Ø: ØA ØØ 87 B4 ØØ Ø2 Ø2 96 ØB C7 B8 ØØ Ø1 Ø1 ØA ØØ
[WD5]   161: 87 BC ØØ Ø2 Ø2 96 ØB C7 C3 ØØ Ø1 Ø1 Ø7 ØØ C7 C8
[1A5]   162: ØØ Ø1 Ø1 Ø3 ØØ 87 CB ØØ Ø2 Ø2 1C ØB C7 DA ØØ Ø1
[UB5]   163: Ø1 Ø9 ØØ C7 EA ØØ Ø1 Ø1 Ø6 ØØ C7 F4 ØØ Ø1 Ø1 Ø5
[XE5]   164: ØØ 6D AØ ØØ Ø4 Ø2 FC Ø3 52 56 E8 ØØ ØØ 5E 5A 59
[ZD5]   165: 58 52 Ø3 16 ØØ ØØ E8 ØØ ØØ 5A E2 CC 89 16 ØØ 5A
[UP5]   166: E8 ØØ ØØ Ø7 5E 9D C3 32 F6 EB E6 FE C2 3A 16 ØØ
[CW5]   167: ØØ 72 DE EB CF 9C 56 57 FC 32 E4 56 AC ØA CØ 74
[AP5]   168: Ø4 FE C4 EB F7 5E 88 65 Ø1 ØA E4 74 12 3A 25 76
[5R5]   169: Ø2 8A 25 8A CC 32 ED 88 65 Ø1 83 C7 Ø2 F3 A4 5F
[LL5]   17Ø: 5E 9D C3 A2 ØØ ØØ 88 26 ØØ ØØ 32 E4 CD 16 8Ø 3E
[EF5]   171: ØØ ØØ ØØ 74 ØA 3C 61 72 Ø6 3C 7A 77 Ø2 24 DF ØB
[XR5]   172: C9 75 Ø1 C3 9C 51 Ø6 8B FE 1E Ø7 FC F2 AE 75 Ø5
[5K5]   173: Ø7 59 9D EB D5 5Ø 8Ø 3E ØØ ØØ ØØ 74 ØD 53 56 A2
[RH5]   174: ØØ ØØ BE ØØ ØØ E8 ØØ ØØ 5E 5B 58 Ø7 59 9D C3 FA
[CR5]   175: Ø6 33 CØ 8E CØ 26 A1 1A Ø4 26 A3 1C Ø4 Ø7 FB C3
[DC5]   176: 9C 51 56 57 55 Ø6 56 1E Ø7 C6 Ø6 ØØ ØØ ØØ 88 26
[F55]   177: ØØ ØØ A2 ØØ ØØ 89 ØE ØØ ØØ 8B 16 ØØ ØØ 89 16 ØØ
[RC5]   178: ØØ 89 3E ØØ ØØ B4 Ø2 CD 16 24 8Ø A2 ØØ ØØ BF ØØ
[GE5]   179: ØØ 57 B9 Ø1 Ø1 FC F3 A4 5E E8 ØØ ØØ 8Ø 3E ØØ ØØ
[7B5]   18Ø: ØØ 74 11 8B 16 ØØ ØØ 2A 16 ØØ ØØ 89 16 ØØ ØØ C6
[9E5]   181: Ø6 ØØ ØØ ØØ 8B EE 3E 8A 46 Ø1 A2 ØØ ØØ 32 E4 Ø5
[YC5]   182: Ø2 ØØ Ø3 E8 32 E4 CD 16 8Ø 3E ØØ ØØ ØØ 74 ØA 3C
[GE5]   183: 61 72 Ø6 3C 7A 77 Ø2 24 DF 83 3E ØØ ØØ ØØ 74 ØE
[3K5]   184: 8B 3E ØØ ØØ 8B ØE ØØ ØØ F2 AE 75 Ø2 EB D6 3C 1B
[VW5]   185: 75 ØA BØ FF 5E Ø7 5D 5F 5E 59 9D C3 3C ØD 75 1C
[SL5]   186: 8A 4C Ø1 32 ED 83 C1 Ø2 5F 57 F3 A4 8B 16 ØØ ØØ
[PF5]   187: 89 16 ØØ ØØ 5E E8 ØØ ØØ 32 CØ EB D9 8Ø FC 52 75
[BD5]   188: Ø7 8Ø 36 ØØ Ø8 8Ø EB 9C 8Ø FC 4B 75 Ø3 E9 ØØ ØØ
[6B5]   189: 8Ø FC 4D 75 Ø3 E9 ØØ ØØ 8Ø FC 48 75 Ø3 E9 ØØ ØØ
[NA5]   19Ø: 8Ø FC 5Ø 75 Ø3 E9 ØØ ØØ 8Ø FC 47 75 Ø3 E9 ØØ ØØ
[7B5]   191: 8Ø FC 4F 75 Ø3 E9 ØØ ØØ 8Ø FC 53 75 Ø3 E9 ØØ ØØ
[YA5]   192: 3D Ø8 ØE 75 Ø3 E9 ØØ ØØ 3D ØØ 77 75 Ø3 E9 ØØ ØØ
[MC5]   193: 3C 2Ø 73 Ø3 E9 ØØ ØØ 8Ø 3E ØØ ØØ ØØ 74 ØA C6 Ø6
[KA5]   194: ØØ ØØ ØØ 5Ø E8 ØØ ØØ 58 8A 1E ØØ ØØ 3A 1C 75 1Ø
[RH5]   195: 56 57 55 Ø6 B8 Ø7 ØE CD 1Ø Ø7 5D 5F 5E E9 ØØ ØØ
[ZF5]   196: E8 ØØ ØØ 3E 88 46 ØØ 8A 44 Ø1 3A Ø4 74 Ø3 FE 44
[EC5]   197: Ø1 8B 16 ØØ ØØ 8Ø 3E ØØ ØØ ØØ 74 ØB FE CA 89 16
[VA5]   198: ØØ ØØ C6 Ø6 ØØ ØØ ØØ 87 16 ØØ ØØ 52 E8 ØØ ØØ 8F
[5B5]   199: Ø6 ØØ ØØ 8B 16 ØØ ØØ 2A 16 ØØ ØØ 89 16 ØØ ØØ C6
[AC5]   2ØØ: Ø6 ØØ ØØ ØØ EB 38 9Ø 89 16 ØØ ØØ 56 57 55 E8 ØØ
[ZB5]   2Ø1: ØØ 5D 5F 5E E9 ØØ ØØ C6 Ø6 ØØ ØØ ØØ 8Ø 3E ØØ ØØ
[WE5]   2Ø2: ØØ 75 Ø3 E9 ØØ ØØ 4D FE ØE ØØ ØØ 8B 16 ØØ ØØ 8Ø
[BW5]   2Ø3: EE Ø1 79 D3 8A 36 ØØ ØØ FE CE FE CA EB C9 C6 Ø6
[HB5]   2Ø4: ØØ ØØ ØØ AØ ØØ ØØ 3A 44 Ø1 75 Ø3 E9 ØØ ØØ 45 FE
[LE5]   2Ø5: Ø6 ØØ ØØ 8B 16 ØØ ØØ FE C6 3A 36 ØØ ØØ 72 A8 32
[FL5]   2Ø6: F6 FE C2 EB A2 C6 Ø6 ØØ ØØ E8 ØØ ØØ ØA CØ 74
[GL5]   2Ø7: ØF FE CA AØ ØØ ØØ 28 Ø6 ØØ ØØ 32 E4 2B E8 EB 87
[175]   2Ø8: E9 ØØ ØØ C6 Ø6 ØØ ØØ AØ ØØ ØØ Ø2 Ø6 ØØ ØØ 72
[SG5]   2Ø9: EF 3A 44 Ø1 77 EA E8 ØØ ØØ FE C2 AØ ØØ ØØ ØØ Ø6
[KD5]   21Ø: ØØ ØØ 32 E4 Ø3 E8 E9 ØØ ØØ C6 Ø6 ØØ ØØ ØØ 8B EE
[AC5]   211: 83 C5 Ø2 C6 Ø6 ØØ ØØ E8 ØØ ØØ E9 ØØ ØØ C6 Ø6
[CH5]   212: ØØ ØØ ØØ 8A 44 Ø1 A2 ØØ ØØ 32 E4 8B E8 Ø3 EE 83
[V85]   213: C5 Ø2 E8 ØØ ØØ E9 ØØ ØØ C6 Ø6 ØØ ØØ ØØ 8Ø 7C Ø1
[VN5]   214: ØØ 75 Ø3 E9 ØØ ØØ 8B FD 8A 4C Ø1 9C 56 8B F7 46
[2L5]   215: FC 2A ØE ØØ ØØ 74 2A Ø6 1E Ø7 32 ED FE C9 74 Ø2
[MD5]   216: F3 A4 BØ 2Ø AA Ø7 5E 56 8B 16 ØØ ØØ 87 16 ØØ ØØ
[VF5]   217: 52 E8 ØØ ØØ 5A 89 16 ØØ ØØ 5E FE 4C Ø1 9D E9 ØØ
[HB5]   218: ØØ 5E 9D E9 ØØ ØØ C6 Ø6 ØØ ØØ ØØ 8Ø 3E ØØ ØØ ØØ
[DK5]   219: 75 Ø3 E9 ØØ ØØ 4D FE ØE ØØ ØØ 8B 16 ØØ ØØ 8Ø EE
[7M5]   22Ø: Ø1 79 Ø8 8A 36 ØØ ØØ FE CE FE CA 89 16 ØØ ØØ 56
[RG5]   221: E8 ØØ ØØ 5E EB 82 C6 Ø6 ØØ ØØ ØØ E8 ØØ ØØ E9 ØØ
[GG5]   222: ØØ 8A 4C Ø1 ØA C9 74 21 3A E1 77 1D ØA E4 74 19
[MV5]   223: ØA CØ 75 Ø6 8A C1 2A C4 FE CØ 8A EC Ø2 E8 FE CD
[7C5]   224: 3A E9 77 Ø5 75 Ø4 28 44 Ø1 C3 9C 5Ø 56 Ø6 1E Ø7
[KP5]   225: 8A 5C Ø1 28 44 Ø1 8B FE 8A D4 32 F6 Ø3 FA 47 8B
[1Z5]   226: F7 8A DØ Ø3 F2 32 FF Ø2 C4 2A D8 FE C3 8B CB FC
[ZL5]   227: F3 A4 Ø7 5E 58 9D C3 8A 44 Ø1 ØA CØ 74 55 ØA E4
[TA5]   228: 74 51 3A 65 C6 9C 42 Ø3 84 Ø3 ØØ Ø2 Ø2 77 Ø2 C4
[LA5]   229: ØC ØØ Ø1 Ø1 Ø3 ØØ 84 ØF ØØ Ø2 Ø2 1C ØB C4 16 ØØ
[R55]   23Ø: Ø1 Ø1 Ø7 ØØ 84 19 ØØ Ø2 Ø2 96 ØB C4 27 ØØ Ø1 Ø1
[SA5]   231: Ø5 ØØ C4 5C ØØ Ø1 Ø1 4C ØØ C4 6Ø ØØ Ø1 Ø1 5C Ø1
[9C5]   232: C4 68 ØØ Ø1 Ø1 4C ØØ C4 9Ø ØØ Ø1 Ø1 5C Ø1 C4 98
[B85]   233: ØØ Ø1 Ø1 5B Ø1 C4 9B ØØ Ø1 Ø1 59 Ø1 84 9E ØØ Ø2
[VD5]   234: Ø2 BE Ø3 C4 C3 ØØ Ø1 Ø1 6Ø Ø1 C4 C8 ØØ Ø1 Ø1 5D
[NA5]   235: Ø1 C4 CB ØØ Ø1 Ø1 4C ØØ C4 CF ØØ Ø1 Ø1 55 Ø1 C4
[1C5]   236: D3 ØØ Ø1 Ø1 Ø7 ØØ C4 D7 ØØ Ø1 Ø1 57 Ø1 C4 DB ØØ
[W85]   237: Ø1 Ø1 4D ØØ C4 E4 ØØ Ø1 Ø1 4F ØØ C4 E7 ØØ Ø1 Ø1
[BC5]   238: 5Ø ØØ 84 F2 ØØ Ø2 Ø2 BE Ø3 C4 F6 ØØ Ø1 Ø1 6Ø Ø1
[L95]   239: C4 FD ØØ Ø1 Ø1 57 Ø1 C5 Ø1 ØØ Ø1 Ø1 6Ø Ø1 C5 Ø5
[S45]   24Ø: ØØ Ø1 Ø1 57 Ø1 C5 Ø9 ØØ Ø1 Ø1 6Ø Ø1 C5 13 ØØ Ø1
[U85]   241: Ø1 54 Ø1 C5 22 ØØ Ø1 Ø1 4C ØØ C5 33 ØØ Ø1 Ø1 55
[C85]   242: Ø1 C5 3A ØØ Ø1 Ø1 4D ØØ C5 3E ØØ Ø1 Ø1 55 Ø1 C5
[995]   243: 66 ØØ Ø1 Ø1 57 Ø1 C5 6A ØØ Ø1 Ø1 Ø7 ØØ 85 6E ØØ
[2A5]   244: Ø2 Ø2 BE Ø3 C5 7B ØØ Ø1 Ø1 4F ØØ 85 86 ØØ Ø2 Ø2
[VB5]   245: 4B Ø6 85 8E ØØ Ø2 Ø2 72 Ø6 85 96 ØØ Ø2 Ø2 99 Ø6
[CF5]   246: 85 9E ØØ Ø2 Ø2 B7 Ø6 85 A6 ØØ Ø2 Ø2 DD Ø6 85 AE
[Y95]   247: ØØ Ø2 Ø2 F2 Ø6 85 B6 ØØ Ø2 Ø2 ØC Ø7 85 BE ØØ Ø2
[AD5]   248: Ø2 5A Ø7 85 C6 ØØ Ø2 Ø2 8A Ø7 85 CD ØØ Ø2 Ø2 18
[8A5]   249: Ø5 C5 D1 ØØ Ø1 Ø1 5D Ø1 C5 D8 ØØ Ø1 Ø1 5D Ø1 85
[GD5]   25Ø: DD ØØ Ø2 Ø2 61 ØC C5 E2 ØØ Ø1 Ø1 54 Ø1 85 F6 ØØ
[595]   251: Ø2 Ø2 18 Ø5 85 F9 ØØ Ø2 Ø2 1C ØC C6 ØB ØØ Ø1 Ø1
[K85]   252: 57 Ø1 C6 ØF ØØ Ø1 Ø1 6Ø Ø1 C6 18 ØØ Ø1 Ø1 57 Ø1
[Z75]   253: C6 1C ØØ Ø1 Ø1 6Ø Ø1 C6 21 ØØ Ø1 Ø1 Ø7 ØØ 86 25
[V85]   254: ØØ Ø2 Ø2 BE Ø3 C6 29 ØØ Ø1 Ø1 Ø7 ØØ C6 2D ØØ Ø1
[185]   255: Ø1 57 Ø1 C6 31 ØØ Ø1 Ø1 6Ø Ø1 C6 35 ØØ Ø1 Ø1 57
[V55]   256: Ø1 C6 39 ØØ Ø1 Ø1 6Ø Ø1 C6 41 ØØ Ø1 Ø1 Ø7 ØØ 86
[X95]   257: 47 ØØ Ø2 Ø2 96 ØB 86 4D ØØ Ø2 Ø2 18 Ø5 C6 51 ØØ
[C75]   258: Ø1 Ø1 5D Ø1 C6 56 ØØ Ø1 Ø1 54 Ø1 86 5C ØØ Ø2 Ø2
[T65]   259: 18 Ø5 C6 61 ØØ Ø1 Ø1 54 Ø1 C6 65 ØØ Ø1 Ø1 Ø7 ØØ
[KE5]   26Ø: C6 6E ØØ Ø1 Ø1 Ø6 ØØ C6 78 ØØ Ø1 Ø1 5D Ø1 C6 7C
[X55]   261: ØØ Ø1 Ø1 54 Ø1 86 84 ØØ Ø2 Ø2 18 Ø5 C6 89 ØØ Ø1
```

Listing 2. Den Hexdump von »system.obj« geben Sie mit SCheck ein

```
[W95]   262: Ø1 54 Ø1 C6 8D ØØ Ø1 Ø1 Ø7 ØØ C6 93 ØØ Ø1 Ø1 Ø6
[1A5]   263: ØØ C6 9F ØØ Ø1 Ø1 5D Ø1 86 A3 ØØ Ø2 Ø2 4D ØC C6
[4B5]   264: AC ØØ Ø1 Ø1 Ø6 ØØ C6 BØ ØØ Ø1 Ø1 54 Ø1 86 B9 ØØ
[G95]   265: Ø2 Ø2 18 Ø5 C6 BD ØØ Ø1 Ø1 5D Ø1 C6 C1 ØØ Ø1 Ø1
[2D5]   266: 54 Ø1 C6 C5 ØØ Ø1 Ø1 Ø6 ØØ 86 CF ØØ Ø2 Ø2 4D ØC
[FD5]   267: C6 D4 ØØ Ø1 Ø1 Ø6 ØØ C6 D8 ØØ Ø1 Ø1 54 Ø1 86 DF
[9B5]   268: ØØ Ø2 Ø2 3B Ø6 C6 E3 ØØ Ø1 Ø1 5D Ø1 C6 ED ØØ Ø1
[LC5]   269: Ø1 54 Ø1 86 F1 ØØ Ø2 Ø2 4D ØC 86 F4 ØØ Ø2 Ø2 3B
[5B5]   27Ø: Ø6 C6 F8 ØØ Ø1 Ø1 5D Ø1 C6 FF ØØ Ø1 Ø1 54 Ø1 87
[NC5]   271: ØB ØØ Ø2 Ø2 4D ØC 87 ØE ØØ Ø2 Ø2 3B Ø6 C7 12 ØØ
[N85]   272: Ø1 Ø1 5D Ø1 87 1C ØØ Ø2 Ø2 18 Ø5 C7 2B ØØ Ø1 Ø1
[L65]   273: 54 Ø1 C7 42 ØØ Ø1 Ø1 57 Ø1 C7 46 ØØ Ø1 Ø1 Ø7 ØØ
[3C5]   274: 87 4A ØØ Ø2 Ø2 BE Ø3 C7 4F ØØ Ø1 Ø1 Ø7 ØØ 87 57
[N75]   275: ØØ Ø2 Ø2 3B Ø6 87 5C ØØ Ø2 Ø2 18 Ø5 C7 6Ø ØØ Ø1
[8A5]   276: Ø1 5D Ø1 C7 65 ØØ Ø1 Ø1 54 Ø1 87 6B ØØ Ø2 Ø2 18
[185]   277: Ø5 C7 7Ø ØØ Ø1 Ø1 54 Ø1 C7 74 ØØ Ø1 Ø1 Ø7 ØØ C7
[Y85]   278: 7D ØØ Ø1 Ø1 Ø6 ØØ C7 85 ØØ Ø1 Ø1 Ø7 ØØ 87 89 ØØ
[H95]   279: Ø2 Ø2 96 ØB C7 9Ø ØØ Ø1 Ø1 5D Ø1 87 94 ØØ Ø2 Ø2
[8C5]   28Ø: 61 ØC 87 97 ØØ Ø2 Ø2 18 Ø5 73 AØ ØØ Ø4 Ø2 F8 Ø7
[GB5]   281: Ø1 77 4C Ø2 45 Ø1 3A Ø5 77 45 9C 5Ø 56 57 Ø6 1E
[5P5]   282: Ø7 8A 4D Ø1 2A CC FE C1 32 ED 8A 5C Ø1 56 57 8B
[ZT5]   283: F7 8A 55 Ø1 32 F6 Ø3 F2 46 32 FF 8B FE Ø3 FB FD
[TU5]   284: F3 A4 5F 5E 83 C6 Ø2 8A 4C FF 32 ED 8A D4 Ø3 FA
[DL5]   285: 47 FC F3 A4 Ø7 5F 5E 8A 44 Ø1 ØØ 45 Ø1 58 9D C3
[KH5]   286: C6 45 Ø1 ØØ 8A 4C Ø1 ØA C9 74 42 ØA E4 74 3E 3A
[CP5]   287: E1 77 3A ØA CØ 75 Ø6 8A C1 2A C4 FE CØ 3A Ø5 76
[DU5]   288: Ø2 8A Ø5 8A EC Ø2 E8 FE CD 3A E9 77 2Ø 9C 5Ø 56
[7H5]   289: 57 Ø6 1E Ø7 88 45 Ø1 83 C7 Ø2 8A C8 32 ED 8A C4
[GP5]   29Ø: 32 E4 Ø3 FØ 46 FC F3 A4 Ø7 5F 5E 58 9D C3 8A 4C
[KG5]   291: Ø1 ØA C9 74 28 8A C1 Ø2 45 Ø1 3A Ø5 77 1F 9C 56
[ØE5]   292: 57 Ø6 1E Ø7 8A 45 Ø1 ØØ 4D Ø1 32 ED 83 C6 Ø2 32
[NU5]   293: E4 Ø3 F8 83 C7 Ø2 FC F3 A4 Ø7 5F 5E 9D C3 8A 4D
[VN5]   294: Ø1 8A 6C Ø1 ØA C9 74 2E 3A CD 77 2A 9C FC 56 57
[ML5]   295: Ø6 1E Ø7 83 C6 Ø2 83 C7 Ø2 2A E9 FE C5 BØ Ø1 51
[ZV5]   296: 56 57 32 ED F3 A6 74 11 5F 5E 59 FE CØ 46 FE CD
[ZR5]   297: 75 ED Ø7 5F 5E 9D 32 CØ C3 5F 5E 59 Ø7 5F 5E 9D
[EN5]   298: C3 8A 4C Ø1 ØA C9 75 Ø1 C3 32 ED 9C 56 FC 83 C6
[6N5]   299: Ø2 AC 3C 61 72 Ø9 3C 7A 77 Ø5 24 DF 88 44 FF E2
[PN5]   3ØØ: FØ 5E 9D C3 8A 4C Ø1 ØA C9 74 1D 56 32 ED 83 C6
[4S5]   3Ø1: Ø2 32 E4 38 Ø4 75 Ø5 FE C4 46 E2 F7 5E ØA E4 74
[9F5]   3Ø2: Ø7 8A C4 B4 Ø1 E9 ØØ ØØ C3 8A 4C Ø1 ØA C9 74 23
[HL5]   3Ø3: 56 32 ED Ø3 F1 46 32 E4 8A D1 38 Ø4 75 Ø7 FE C4
[TW5]   3Ø4: 4E FE CA E2 F5 FE C2 5E ØA E4 74 Ø7 8A C4 8A E2
[DF5]   3Ø5: E9 ØØ ØØ C3 C6 44 Ø1 ØØ 26 8A ØD ØA C9 74 21 3A
[VH5]   3Ø6: ØC 77 1D 9C 56 57 1E Ø6 1E Ø6 1F Ø7 47 26 88 4C
[ZR5]   3Ø7: Ø1 83 C6 Ø2 87 F7 FC 32 ED F3 A4 Ø7 1F 5F 5E 9D
[ML5]   3Ø8: C3 Ø3 C3 13 D1 C3 2B C3 1B D1 C3 56 57 52 5Ø F7
[DW5]   3Ø9: E3 8B FA 8B FØ 58 F7 E1 Ø3 F8 58 F7 E3 Ø3 F8 8B
[SR5]   31Ø: D7 8B C6 5F 5E C3 53 ØB D9 5B 75 Ø2 F9 C3 55 56
[LS5]   311: 57 8B F3 8B F9 B9 2Ø ØØ 33 DB 33 ED D1 EØ D1 D2
[SW5]   312: D1 D3 D1 D5 53 55 2B DE 1B EF 73 Ø6 5D 5B F8 EB
[VS5]   313: Ø5 9Ø 83 C4 Ø4 F9 D1 DØ D1 D2 E2 E4 8B CD 5F 5E
[SP5]   314: 5D F8 C3 F7 DØ F7 D2 Ø5 Ø1 ØØ 83 D2 ØØ C3 8Ø FB
[VB5]   315: Ø2 72 Ø5 8Ø FB 1Ø 76 Ø2 F9 C3 5Ø 53 55 52 56 33
[WD5]   316: ED 88 1E ØØ ØØ 33 C9 32 FF 8A 1E ØØ ØØ E8 ØØ ØØ
[DE5]   317: 8Ø C3 3Ø 8Ø FB 39 76 Ø3 8Ø C3 Ø7 53 45 ØB CØ 75
[MS5]   318: E8 ØB D2 75 E4 8B CD 3A ØC 77 13 88 4C Ø1 83 C6
[HN5]   319: Ø2 58 88 Ø4 46 E2 FA 5E 5A 5D 5B 58 F8 C3 58 E2
[BN5]   32Ø: FD 5E C6 44 Ø1 ØØ 5A 5D 5B 58 F9 C3 C6 Ø6 ØØ ØØ
[NF5]   321: ØØ 33 CØ 33 D2 8A 4C Ø1 ØA C9 75 Ø1 C3 53 51 8B
[BT5]   322: FE 32 ED Ø3 F9 47 8A 1D 8Ø E3 DF 8Ø FB 48 75 ØB
[7N5]   323: B7 1Ø FE C9 8Ø F9 Ø8 76 38 EB 1Ø 8Ø FB 42 75 ØF
[XS5]   324: B7 Ø2 8Ø F9 21 77 Ø4 FE C9 EB 26 59 5B F9 C3 B7
[VR5]   325: ØA 8Ø F9 ØB 77 F5 8Ø 7C Ø2 2D 75 1Ø FE C9 B3 ØA
[FL5]   326: 8B FE 83 C7 Ø3 C6 Ø6 ØØ ØØ FF EB ØC 8Ø F9 ØB 74
[FT5]   327: DA 8A DF 8B FE 83 C7 Ø2 53 51 33 C9 32 FF E8 ØØ
[1P5]   328: ØØ 59 5B 8A 3D 8Ø FF 61 72 Ø3 8Ø E7 DF 8Ø EF 3Ø
[UM5]   329: 8Ø FF Ø9 76 Ø3 8Ø EF Ø7 53 8A DF 32 FF Ø3 C3 83
[MB5]   33Ø: D2 ØØ 5B 47 E2 D2 8Ø 3E ØØ ØØ ØØ 74 Ø3 E8 ØØ ØØ
[ØR5]   331: 59 5B F8 C3 5Ø 51 52 32 F6 8B C2 B1 Ø7 D3 EØ 8B
[DT5]   332: F8 Ø3 FA Ø3 FA Ø3 FA Ø3 FA Ø3 FA Ø3 FA Ø3 FA Ø3
[8U5]   333: FA Ø3 FA Ø3 FA Ø3 FA Ø3 FA Ø3 FA Ø3 FA Ø3 FA Ø3
[8U5]   334: FA Ø3 FA Ø3 FA Ø3 FA Ø3 FA Ø3 FA Ø3 FA Ø3 FA Ø3
[8U5]   335: FA Ø3 FA Ø3 FA Ø3 FA Ø3 FA Ø3 FA Ø3 FA Ø3 FA Ø3
[RL5]   336: FA 5A 8A C6 32 E4 D1 EØ Ø3 F8 59 58 8E Ø6 ØØ ØØ
[MK5]   337: C3 9C Ø6 1E Ø7 FC AØ ØØ ØØ 32 E4 B1 Ø3 D3 EØ Ø5
[5M5]   338: ØØ ØØ 8B FØ BF ØØ ØØ B9 Ø4 ØØ F3 A5 Ø7 9D 8Ø 3E
[RG5]   339: ØØ ØØ ØØ 74 ØA B4 Ø2 32 FF BA ØØ 19 CD 1Ø C3 A1
[5H5]   34Ø: ØØ ØØ 8B DØ FE C4 3A 26 ØØ ØØ 76 Ø6 32 F6 FE C2
[3U5]   341: FE CØ FE CØ 8A ØE ØØ ØØ 3A C1 76 17 8A D1 FE CA
[5N5]   342: 2A C1 FE C8 74 ØD 8A C8 32 ED 52 51 E8 ØØ ØØ 59
[AF5]   343: E2 F9 5A 89 16 ØØ ØØ Ø3 16 ØØ ØØ 86 F2 32 FF B4
[ZG5]   344: Ø2 CD 1Ø C3 9C 56 1E Ø6 FC 8B 14 86 4Ø 9C 94 ØØ
[R85]   345: 85 56 ØØ Ø2 Ø2 95 Ø7 85 81 ØØ Ø2 Ø2 95 Ø7 C6 33
[F85]   346: ØØ Ø1 Ø1 5E Ø1 C6 3B ØØ Ø1 Ø1 5E Ø1 86 3E ØØ Ø2
[HH5]   347: Ø2 CE Ø9 C6 7E ØØ Ø1 Ø1 5F Ø1 C6 D7 ØØ Ø1 Ø1 5F
[LC5]   348: Ø1 86 EF ØØ Ø2 Ø2 B3 Ø9 C7 18 ØØ Ø1 Ø1 5F Ø1 87
[EC5]   349: 1E ØØ Ø2 Ø2 ØB ØA C7 7E ØØ Ø1 Ø1 Ø1 ØØ C7 87 ØØ
[375]   35Ø: Ø1 Ø1 ØB ØØ C7 9Ø ØØ Ø1 Ø1 ØC ØØ C7 95 ØØ Ø1 Ø1
[B95]   351: Ø3 ØØ C7 AØ ØØ Ø1 Ø1 ØA ØØ C7 BØ ØØ Ø1 Ø1 Ø7 ØØ
[4D5]   352: C7 B8 ØØ Ø1 Ø1 Ø6 ØØ C7 C6 ØØ Ø1 Ø1 Ø5 ØØ 87 DD
[985]   353: ØØ Ø2 Ø2 77 Ø2 C7 E5 ØØ Ø1 Ø1 Ø7 ØØ C7 E9 ØØ Ø1
[EH5]   354: Ø1 Ø3 ØØ B9 AØ B8 ØØ Ø2 F4 ØB F2 E8 ØØ ØØ 8B 4C
[YM5]   355: Ø2 86 E9 8B 74 Ø8 87 F7 FC 1E Ø6 1F Ø7 51 56 32
[9T5]   356: ED F3 A5 5E 81 C6 AØ ØØ 59 FE CD 75 FØ Ø7 1F 5E
[RC5]   357: 9D C3 9C 5Ø 53 51 52 56 57 8Ø 3E ØØ ØØ ØØ 74 1B
[HE5]   358: AØ ØØ ØØ 3A 44 Ø1 74 13 8A Ø4 32 E4 Ø3 FØ 46 8B
[ØV5]   359: FE 47 8B C8 2A ØE ØØ ØØ FD F3 A4 5F 5E 5A 59 5B
[SG5]   36Ø: 58 9D C3 51 AØ ØØ ØØ 32 E4 8A ØE ØØ ØØ F6 F1 8B
[UG5]   361: 16 ØØ ØØ Ø3 DØ 59 C3 8A 4C Ø1 ØA C9 75 Ø1 C3 32
[HT5]   362: ED 8B FE 83 C7 Ø2 BØ 2Ø Ø6 1E Ø7 FC F3 AA Ø7 8B
[RB5]   363: 16 ØØ ØØ 87 16 ØØ ØØ 52 E8 ØØ ØØ 8F Ø6 ØØ ØØ 8B
[GA5]   364: EE 83 C5 Ø2 C6 Ø6 ØØ ØØ ØØ C6 44 Ø1 ØØ E8 ØØ ØØ
[YH5]   365: 89 16 ØØ ØØ 56 57 55 E8 ØØ ØØ 5D 5F 5E C3 5C 9C
[995]   366: 6A ØØ 84 Ø2 ØØ Ø2 Ø2 1C ØB C4 31 ØØ Ø1 Ø1 4F ØØ
[6C5]   367: C4 37 ØØ Ø1 Ø1 54 Ø1 C4 4C ØØ Ø1 Ø1 54 Ø1 C4 5B
[E45]   368: ØØ Ø1 Ø1 54 Ø1 C4 61 ØØ Ø1 Ø1 Ø6 ØØ C4 67 ØØ Ø1
[UA5]   369: Ø1 57 Ø1 C4 87 ØØ Ø1 Ø1 57 Ø1 C4 8B ØØ Ø1 Ø1 Ø7
[S95]   37Ø: ØØ 84 8F ØØ Ø2 Ø2 BE Ø3 C4 93 ØØ Ø1 Ø1 Ø7 ØØ C4
[HD5]   371: 9C ØØ Ø1 Ø1 54 Ø1 84 A4 ØØ Ø2 Ø2 4D ØC C4 A8 ØØ
[575]   372: Ø1 Ø1 Ø7 ØØ 84 AE ØØ Ø2 Ø2 96 ØB 65 8A Ø2 ØØ ØØ
[A8Ø]   373: 74
```

Listing 2. Der Hexdump von »system.obj«

```
[YN3]     1: ;Datei    : beispiel.asm
[YL5]     2: ;Funktion: Demonstration einiger
[UC4]     3: ;          Bibliotheksroutinen
[GK5]     4: ;Sprache:  MASM ab Version 4.Ø, TASM
[AX3]     5: ;Autor:    Ingo Kubbilun
[7X4]     6: ;(c) 1991 DMV Widuch GmbH & Co. KG
[ØØØ]     7:
[8F4]     8: Data       Segment Word Public
[ØØØ]     9:
[DV2]    1Ø: ;Frame-Struktur
[EB5]    11: win1       db 1Ø              ;x-Position 1Ø
[EW3]    12:            db 1Ø              ;y-Position 1Ø
[3Z2]    13:            db 1Ø              ;Höhe 1Ø
[G43]    14:            db 27              ;Breite 27
[W84]    15:            db 1Eh             ;Farbattribut
[VG3]    16:            db 1               ;reserviert
[RG4]    17:            dw offset codes1   ;Zeiger auf
[XZ1]    18:                               ;ASCII-Codes
[WB4]    19:            dw offset back1    ;Zeiger auf
[YA1]    2Ø:                               ;leeren
[VA1]    21:                               ;Bereich
[ØØØ]    22:
[MA4]    23: codes1     db 218, 25 dup (196), 191
[5X2]    24:            db 179, 25 dup (32), 179
[DØ3]    25:            db 192, 25 dup (196), 217
[ØØØ]    26:
[ØB3]    27: back1      db 27*2*1Ø dup (?)
[ØØØ]    28:
[DV2]    29: ;Frame-Struktur
[FU1]    3Ø: win2       db 5
[3GØ]    31:            db 5
[1LØ]    32:            db 11
[DLØ]    33:            db 36
[6VØ]    34:            db 4fh
[ZFØ]    35:            db 1
[3M2]    36:            dw offset codes2
[BE2]    37:            dw offset back2
[ØØØ]    38:
[DM4]    39: codes2 db 218, 32 dup (196), 191, 1, 1
[NF5]    4Ø:        db 179,"     Demonstrationsfenster"
[Z32]    41:        db ":     ",179,2,2
[VD3]    42:        db 195, 32 dup (196),18Ø,2,2
[CN3]    43:        db 179, " Das Wort ", Ø, 4eh
[374]    44:        db "Hallo", Ø, 4fh, " ist gelb "
[YH2]    45:        db "ge-    ", 179, 2, 2
[H93]    46:        db 179, " schrieben..."
[TY2]    47:        db 19 dup (32), 179, 2, 2
[ØH3]    48:        db 179, 32 dup (32), 179, 2, 2
[ØH3]    49:        db 179, 32 dup (32), 179, 2, 2
[ØH3]    5Ø:        db 179, 32 dup (32), 179, 2, 2
[ØH3]    51:        db 179, 32 dup (32), 179, 2, 2
[WN3]    52:        db 192, 32 dup (196), 217, 2, 2
[RØ2]    53:        db 1,1,4Ø dup (2)
[ØØØ]    54:
[7B3]    55: back2      db 36*2*11 dup (?)
[ØØØ]    56:
[474]    57: parstr     db 127,Ø,127 dup (?)
[ØØØ]    58:
[756]    59: msg        db 2Ø,2Ø,Ødh,Øah,"Taste drücken"
[412]    6Ø:            db "...",Ødh,Øah
[Ø56]    61: wort1      db 57,57,"Sie haben diesem Prog"
[5D5]    62:            db "ramm folgende Parameter übe"
[DC2]    63:            db "rgeben: ",22h
[R44]    64: wort2      db 4,4,22h,Ødh,Øah,Øah
[KD6]    65: wort3      db 29,29,"Keine Parameter überg"
[YZ2]    66:            db "eben!",Ødh,Øah,Øah
[E86]    67: wort4      db 35,35,"Dies ist ein virtuell"
[BW2]    68:            db "er Bildschirm."
[RØ6]    69: wort5      db 31,31,"Dies ist ein einfache"
[L92]    7Ø:            db "s Fenster."
[T14]    71: wort6      db Ø8h,Ø8h," -Test- "
[356]    72: wort7      db 31,31,"Dies ist ein komplexe"
[L92]    73:            db "s Fenster."
[AA6]    74: wort8      db 58,58,"Nun wurde ein virtuel"
[S35]    75:            db "ler Bild-schirm in das Fens"
[5G2]    76:            db "ter gelegt."
[XØ6]    77: wort9      db 39,39,"Probieren Sie die Ein"
[VR4]    78:            db "gaberou- tine aus: ",Ødh,Øah
[3K3]    79: instr      db 9Ø,Ø,9Ø dup (?)
[ØØØ]    8Ø:
[RF3]    81:            include sysdata.inc
[ØØØ]    82:
[N12]    83: Data       ends
[ØØØ]    84:
[XE4]    85: Code       segment byte public
[7Z3]    86:            Assume  CS:Code, DS:Data
[ØØØ]    87:
[6B3]    88: Init:      mov     AX,Data
[1V1]    89:            mov     DS,AX
[6N2]    9Ø:            call    DetectScr
[GD2]    91:            call    InitScr
[VX1]    92:            mov     AH,62h
[3K1]    93:            int     21h
[4V1]    94:            mov     ES,BX
[2Y1]    95:            mov     DI,8Øh
[PV3]    96:            mov     SI,offset parstr
[XH2]    97:            call    ParToStr
[KY3]    98:            cmp     byte ptr [SI+1],Ø
[UB2]    99:            jz      NoParams
[5P3]   1ØØ:            mov     SI,offset wort1
[NX1]   1Ø1:            call    Print
```

```
[PV3]  1Ø2:             mov     SI,offset parstr
[NX1]  1Ø3:             call    Print
[9P3]  1Ø4:             mov     SI,offset wort2
[NX1]  1Ø5:             call    Print
[SU2]  1Ø6:             jmp     short next
[575]  1Ø7: NoParams:   mov     SI,offset wort3
[NX1]  1Ø8:             call    Print
[GN3]  1Ø9: Next:       call    WaitKey
[ZX1]  11Ø:             mov     AL,2Øh
[642]  111:             call    ClrScr
[U72]  112:             mov     AX,Ø11Eh
[D82]  113:             mov     DX,ØAØAh
[U82]  114:             mov     CX,3CØCh
[MK2]  115:             call    DefWindow
[UL1]  116:             mov     AH,1
[AM2]  117:             call    SelWindow
[ZX1]  118:             mov     AL,2Øh
[642]  119:             call    ClrScr
[HP3]  12Ø:             mov     SI,offset wort4
[NX1]  121:             call    Print
[RD2]  122:             call    WaitKey
[ØT1]  123:             xor     AH,AH
[AM2]  124:             call    SelWindow
[TW1]  125:             mov     AL,177
[642]  126:             call    ClrScr
[AC3]  127:             mov     SI,offset win1
[UG2]  128:             call    PutFrame
[MZ1]  129:             mov     DX,16h
[U62]  13Ø:             call    GotoXY
[NP3]  131:             mov     SI,offset wort5
[NX1]  132:             call    Print
[RD2]  133:             call    WaitKey
[AC3]  134:             mov     SI,offset win1
[KD2]  135:             call    DelFrame
[8Z1]  136:             mov     AL,1Eh
[XG2]  137:             call    SetColor
[ZX1]  138:             mov     AL,2Øh
[642]  139:             call    ClrScr
[ZW1]  14Ø:             mov     CX,2ØØ
[TP3]  141:             mov     SI,offset wort6
[BR2]  142: loop:       push    CX
[NX1]  143:             call    Print
[VD1]  144:             pop     CX
[2V1]  145:             loop    loop
[DC3]  146:             mov     SI,offset win2
[UG2]  147:             call    PutFrame
[MZ1]  148:             mov     DX,16h
[U62]  149:             call    GotoXY
[XP3]  15Ø:             mov     SI,offset wort7
[NX1]  151:             call    Print
[RD2]  152:             call    WaitKey
[782]  153:             mov     AX,Ø14Fh
[N62]  154:             mov     DX,Ø7Ø8h
[872]  155:             mov     CX,1EØ6h
[MK2]  156:             call    DefWindow
[UL1]  157:             mov     AH,1
[AM2]  158:             call    SelWindow
[ZX1]  159:             mov     AL,2Øh
[642]  16Ø:             call    ClrScr
[1R3]  161:             mov     SI,offset wort8
[NX1]  162:             call    Print
[RD2]  163:             call    WaitKey
[ZX1]  164:             mov     AL,2Øh
[642]  165:             call    ClrScr
[5R3]  166:             mov     SI,offset wort9
[NX1]  167:             call    Print
[KS3]  168:             mov     SI,offset instr
[GV1]  169:             xor     AX,AX
[PV1]  17Ø:             xor     CX,CX
[YX1]  171:             call    Input
[ØT1]  172:             xor     AH,AH
[AM2]  173:             call    SelWindow
[CM1]  174:             mov     AL,7
[XG2]  175:             call    SetColor
[ZX1]  176:             mov     AL,2Øh
[642]  177:             call    ClrScr
[B62]  178:             mov     AX,4CØØh
[3K1]  179:             int     21h
[ØØØ]  18Ø:
[2G3]  181: WaitKey     proc    near
[B63]  182:             mov     SI,offset msg
[NX1]  183:             call    Print
[CX1]  184:             call    Flush
[ØT1]  185:             xor     AH,AH
[GK1]  186:             int     16h
[3PØ]  187:             ret
[ZE2]  188: WaitKey     endp
[ØØØ]  189:
[KF3]  19Ø:             include syscode.inc
[ØØØ]  191:
[112]  192: Code        ends
[ØØØ]  193:
[RF3]  194: Stack       Segment Stack
[782]  195:             dw 2ØØh dup (?)
[S42]  196: Stack       ends
[ØØØ]  197:
[SP1]  198:             end     Init
```

Listing 3. Das Beispielprogramm »beispiel.asm« verdeutlicht Ihnen den Gebrauch der Bibliotheksroutinen

```
[HZ2]    1: ;Datei:    leer.asm
[1B6]    2: ;Funktion: Programmgerüst, daß die
[5A5]    3: ;          Bibliothek "system" braucht
[TL5]    4: ;Sprache:  MASM ab Version 4.Ø,  TASM
[AX3]    5: ;Autor:    Ingo Kubbilun
[7X4]    6: ;(c) 1991 DMV Widuch GmbH & Co. KG
[ØØØ]    7:
[ØØØ]    8:
[8F4]    9: Data       segment word public
[ØØØ]   1Ø:
[DN4]   11: ; Hier bringen Sie Ihre Daten-
[7B3]   12: ; definitionen unter
[ØØØ]   13:
[RF3]   14:           include sysdata.inc
[ØØØ]   15:
[N12]   16: Data      ends
[ØØØ]   17:
[XE4]   18: Code      segment byte public
[7Z3]   19:           assume  CS:Code, DS:Data
[ØØØ]   2Ø:
[6B3]   21: Init:     mov     AX,Data
[1V1]   22:           mov     DS,AX
[3V1]   23:           mov     ES,AX
[6N2]   24:           call    DetectScr
[GD2]   25:           call    InitScr
[ØØØ]   26:
[H35]   27: ; An dieser Stelle beginnt Ihre
[GV2]   28: ; Programmierung
[ØØØ]   29:
[KF3]   3Ø:           include syscode.inc
[ØØØ]   31:
[112]   32: Code      ends
[ØØØ]   33:
[RF3]   34: Stack     segment stack
[782]   35:           dw 2ØØh dup (?)
[S42]   36: Stack     ends
[ØØØ]   37:
[SP1]   38:           end     Init
```

Listing 4. »leer.asm« benutzen Sie bei der Anfertigung eigener Programme

```
[X93]    1: ;Datei:    syscode.inc
[LØ6]    2: ;Funktion: Include-Datei, die System-
[3K4]    3: ;          Prozeduren als extern
[3M2]    4: ;          deklariert
[GK5]    5: ;Sprache:  MASM ab Version 4.Ø, TASM
[AX3]    6: ;Autor:    Ingo Kubbilun
[7X4]    7: ;(c) 1991 DMV Widuch GmbH & Co. KG
[ØØØ]    8:
[WØ6]    9: extrn DetectScr: near, InitScr: near
[YA5]   1Ø: extrn ClrScr: near, ClrEol: near
[TM5]   11: extrn DelLine: near, InsLine: near
[646]   12: extrn SetColor: near, GetColor: near
[BF6]   13: extrn DefWindow: near, AskWindow: near
[W96]   14: extrn SelWindow: near, PutFrame: near
[136]   15: extrn DelFrame: near, ScrollUp: near
[E46]   16: extrn ScrollDn: near, ScrollLf: near
[8U5]   17: extrn ScrollRg: near, GotoXY: near
[A15]   18: extrn GetXY: near, Print: near
[1A6]   19: extrn ConvString: near, ReadKey: near
[YZ4]   2Ø: extrn Flush: near, Input: near
[WG6]   21: extrn DeleteStr: near, InsertStr: near
[E56]   22: extrn CopyStr:Near, ConcatStr: near
[8C5]   23: extrn PosStr: near, UpStr: near
[GL5]   24: extrn IgnChar: near, CutChar: near
[DB5]   25: extrn ParToStr: near, DAdd: near
[1D6]   26: extrn DSub: near, DMul: near, DDiv: near
[7C5]   27: extrn DNeg: near, NumToStr: near
[MK3]   28: extrn StrToNum: near
```

Listing 5. »syscode.inc« bindet alle Routinen in den Quelltext ein

```
[D93]    1: ;Datei    : sysdata.inc
[P66]    2: ;Funktion: Include-Datei, die Variable
[7U4]    3: ;          von "system" als extern
[3M2]    4: ;          deklariert
[GK5]    5: ;Sprache:  MASM ab Version 4.Ø, TASM
[AX3]    6: ;Autor:    Ingo Kubbilun
[7X4]    7: ;(c) 1991 DMV Widuch GmbH & Co. KG
[ØØØ]    8:
[FL5]    9: extrn ScrMode: byte, ScrSeg: word
[1X5]   1Ø: extrn WinYPos: byte, WinXPos: byte
[V45]   11: extrn Height: byte, Width: byte
[HT4]   12: extrn ActY: byte, ActX: byte
[GR5]   13: extrn WinAttr: byte, CurFlag: byte
[HT5]   14: extrn WSelect: byte, Windows: byte
[DK5]   15: extrn KeyFlag: byte, BufAdr: word
[3G5]   16: extrn Insert: byte, StrBuf: byte
[9N5]   17: extrn BufPos: byte, AnzKeys: word
[YE5]   18: extrn PosXY: word, SysStr: byte
[BL5]   19: extrn OutFlag: byte, DelFlag: byte
[ØB5]   2Ø: extrn Basis: byte, NegFlag: byte
[1R3]   21: extrn ScrUpFlag: byte
```

Listing 6. Die Include-Datei »sysdata.inc« stellt die internen Variablen der Bibliothek zur Verfügung

Befehlsreferenz

Die Befehle der Intel-Prozessoren

Moderne Assembler erlauben ein fast komfortables Programmieren. Allerdings lassen sich die Befehle und Register schwer einprägen. Einen guten Überblick über die Kommandos der einzelnen Prozessoren geben nachfolgende Referenztabellen. Das erleichtert Ihnen die Programmierung, und Sie sparen Zeit, die Sie sonst für das Nachschlagen in Handbüchern und Assembler-Abhandlungen aufwenden müßten.

Die meisten Programmierer lösen ihre PC-Aufgaben in höheren Programmiersprachen, doch letztendlich erhalten die PC-Prozessoren ausschließlich Anweisungen in Maschinencode.

Wenn Sie direkt in Maschinensprache programmieren, können Sie viel Zeit sparen, weil der Übersetzungsvorgang vor oder während des Programmablaufs entfällt. Doch Maschinencode besteht nur aus endlosen Zahlenkolonnen, die für Sie völlig unübersichtlich sind.

Aus diesem Grund haben sich die Entwickler von Mikroprozessoren zu allen Maschinencode-Kommandos kurze, aber verständliche Bezeichnungen einfallen lassen (beispielsweise »sub« für »subtrahieren«, »out« für »ausgeben« oder »call« für »aufrufen«). Diese Kurzbezeichnungen – auch Assembler-Befehle genannt – vereinfachen den Umgang mit den Prozessoren erheblich. Um die Assembler-Kommandos wieder in Maschinencode umzusetzen, ist der Einsatz des Assemblers erforderlich.

Tabelle 1 listet alle Befehle der Mikroprozessoren mit ihrer deutschen Übersetzung auf. Tabelle 2 erläutert Ihnen die grundlegenden Funktionen der alphabetisch sortierten Kommandos. Tabelle 3 führt die Assembler-Befehle numerisch sortiert auf. Sie können mit dieser Tabelle kurze Maschinencode-Listings schnell von Hand übersetzen, wenn Ihnen gerade kein Disassembler zur Verfügung steht. Die Tabellen 4 bis 6 erläutern die in den Tabellen 2 und 3 benutzten variablen Werte. Ab dem 80386 erhalten Sie je nach Wunsch Zugriff auf einen größeren Adreß- und Datenbus. In diesem Fall ändern sich die angegebenen Operanden und Sie erhalten die in Tabelle 7 dargestellten Werte. Beachten Sie aber, daß ab dem 80386 zwei Präfixe, das Operand-Size- und das Adress-Size-Präfix, die aktuelle Auswahl der Operanden- und der effektiven Adreßgröße überschreiben können. Die Anwesenheit der Operand-Size- und des Adress-Size-Präfixes zwingen die Operanden- und die effektive Adreßgröße auf den Wert, der der voreingestellten Belegung entgegengesetzt ist. Ist die Operandengröße beispielsweise für 32-Bit-Datenoperationen vorgesehen, zwingt der Operand-Size-Präfix das Kommando zu 16-Bit-Operationen. Ist die voreingestellte effektive Adreßgröße gleich 16 Bit, weist der Adress-Size-Präfix den Prozessor an, mit 32-Bit-Adressen zu arbeiten.

Diese 32-Bit-Erweiterung ist in allen 80386-Modi einschließlich des Real-Adress- und des virtuellen 8086-Modus eingerichtet. In diese Modi ist die Standarddefinition 16 Bit groß. Die Präfixe benutzen Sie daher, um die 32-Bit-Operanden oder -Adressen zu bestimmen.

Wenn die Tabellen nicht ausdrücklich darauf hinweisen, beeinträchtigen Kommandos mit 8- oder 16-Bit-Operanden nicht die Inhalte der höherwertigen Bits des erweiterten Registers.

Mit diesen Tabellen sind Sie nun gut für Ihre nächste Programmierung in Assembler gewappnet.

Denn wer kann schon von sich behaupten, alle Assemblerbefehle und ihre Verfügbarkeit auf den unterschiedlichen Intel-Prozessoren auswendig zu kennen.

(Markus Zietlow/Kerrin Uhlmann/db)

– Stephen Fedtke: Vom Prozessorkonzept zur Programmierung, Vieweg, ISBN 3-528-04623-6

– Thom Hogan: Die PC-Referenz für Programmierer, Systhema, ISBN 3-89390-250-3

– Ross P. Nelson: Programmierhandbuch 386, Microsoft Press/Vieweg, ISBN 3-528-04693-7

Die Assembler-Befehle in Englisch und Deutsch

Befehl (Abkürzung)	englische Intel-Spezifikation	deutsche Bedeutung
aaa	ASCII Adjust after Addition	ASCII-Korrektur nach Addition
aad	ASCII Adjust AX before Division	ASCII-Korrektur vor Division
aam	ASCII Adjust AX after Multiply	ASCII-Korrektur nach Multiplikation
aas	ASCII Adjust AL after Subtraction	ASCII-Korrektur nach Subtraktion
adc	Add with Carry	Addition mit Übertrag
add	Add	Ganzahlen-Addition
and	Logical And	Verknüpfung durch logisches Und
arpl	Adjust RPL Field of Selector	Korrektur des RPL-Feldes eines Selektors
bound	Check Array Index Against Bounds	Grenzwerttest
bsf	Bit Scan Forward	Bit-Abtastung vorwärts
bsr	Bit Scan Reverse	Bit-Abtastung rückwärts
bswap	Byte Swap	Byte-Austausch
bt	Bit Test	Bit-Test
btc	Bit Test and Complement	Bit testen und komplementieren
btr	Bit Test and Reset	Bit testen und löschen
bts	Bit Test and Set	Bit testen und setzen
call	Call Procedure	Unterprogramm aufrufen
cbw	Convert Byte to Word	Byte in Wort umwandeln
cdq	Convert Doubleword to Quadword	Doppelwort in Vierfachwort umwandeln
clc	Clear Carry Flag	Übertragskennzeichen löschen
cld	Clear Direction Flag	Richtungskennzeichen löschen

Befehl (Abkürzung)	englische Intel-Spezifikation	deutsche Bedeutung
clts	Clear Task-Switched Flag in CR0	TS-Kennzeichen löschen
cmc	Complement Carry Flag	Übertragskennzeichen komplementieren
cmp	Compare Two Operands	Vergleich zweier Operanden
cmps	Compare String Operands	Vergleich von Zeichenketten-Operanden
cmpxchg	Compare and Exchange	Vergleich und Werteaustausch
cwd	Convert Word to Doubleword	Wort in Doppelwort umwandeln
daa	Decimal Adjust after Addition	Dezimalkorrektur nach Addition
das	Decimal Adjust AL after Subtraction	Dezimalkorrektur nach Subtraktion
dec	Decrement by 1	Operand um 1 dekrementieren
div	Unsigned Divide	Division ohne Vorzeichen
enter	Make Stack Frame for Procedure Parameters	Stapel-Frame anlegen
hlt	Halt	Prozessor-Halt
idiv	Signed Divide	Division mit Vorzeichen
imul	Signed Multiply	Multiplikation mit Vorzeichen
in	Input from Port	Eingabe über Ein- Ausgabeport
inc	Increment by 1	Operand um 1 inkrementieren
ins	Input from Port to String	Eingabe einer Zeichenkette über einen Port
int	Call to Interrupt Procedure	Software-Interrupt
into	Call to Interrupt Procedure on Overflow	Software-Interrupt 4 abhängig vom Übertragskennzeichen
invd	Invalidate Cache	Cache löschen
invlpg	Invalidate TLB Entry	TLB-Eintrag löschen
iret	Interrupt Return	Rückkehr aus Interrupt
j*cc*	Jump if Condition is Met	Verzweigung, wenn Bedingung *cc* erfüllt ist
jmp	Jump	Verzweigung
lahf	Load Flags into AH Register	Kopieren der Flag-Werte ins AH-Register
lar	Load Access Rights Byte	Zugriffsrechte laden
l*seg*	Load Full Pointer	Laden des Segmentregisters *seg*
lea	Load Effective Adress	Laden der effektiven Adresse
leave	High Level Procedure Exit	Auflösen des Stapel-Frames
lgdt	Load Global Descriptor Table Register	GDT-Register laden
lidt	Load Interrupt Descriptor Table Register	IDT-Register laden
lldt	Load Local Descriptor Table Register	LDT-Register laden
lmsw	Load Machine Status Word	Maschinenstatuswort laden
lock	Assert LOCK# Signal Prefix	Bussignal LOCK# aktivieren
lods	Load String Operand	Zeichenkette laden
loop	Loop Control with CX Counter	Schleifenkontrolle
loop*cc*	Conditional Loop Control with CX Counter	Schleifenkontrolle (bedingt)
lsl	Load Segment Limit	Segmentgrenze laden
ltr	Load Task Register	Taskregister laden
mov	Move Data	Daten im Speicher oder in Registern ablegen

Befehl (Abkürzung)	englische Intel-Spezifikation	deutsche Bedeutung
movs	Move Data from String to String	Zeichenketten verschieben
movsx	Move with Sign-Extend	Übertragung mit Vorzeichen-Erweiterung
movzx	Move with Zero-Extend	Übertragung mit Null-Erweiterung
mul	Unsigned Multiplication of AL or AX	Multiplikation ohne Vorzeichen
neg	Two's Complement Negation	Negation
nop	No Operation	Nulloperation
not	One's Complement Negation	Komplement bilden
or	Logical Inclusive OR	Verknüpfung durch logisches Oder
out	Output to Port	Ausgabe über Ein- Ausgabeport
outs	Output String to Port	Ausgabe einer Zeichenkette über einen Port
pop	Pop a Word from the Stack	Daten vom Stapel holen
popa	Pop all General Registers	16-Bit-Register vom Stapel holen
popad	Pop all General Registers	32-Bit-Register vom Stapel holen
popf	Pop Stack into FLAGS Register	Kennzeichenregister vom Stapel holen
popfd	Pop Stack into EFLAGS Register	Kennzeichenregister des erweiterten Registersatzes vom Stapel holen
shr	Shift Instruction	Rechtsrotation (logisch)
shrd	Double Precision	Rechtsrotation (doppelt)
pusha	Push all General Registers	16-Bit-Register auf den Stapel legen
pushad	Push all General Registers	32-Bit-Register auf den Stapel legen
pushf	Push Flags Register onto the stack	Kennzeichenregister auf den Stapel legen
pushfd	Push Flags Register onto the Stack	Kennzeichenregister des erweiterten Registersatzes auf Stapel legen
rcl	Rotate	Linksrotation durch das Übertragskennzeichen
rcr	Rotate	Rechtsrotation durch das Übertragskennzeichen
rep	Repeat Following String Operation	Wiederholungspräfix für Zeichenkettenkommandos
ret	Return from Procedure	Rückkehr aus einem Unterprogramm
rol	Rotate	Linksrotation
ror	Rotate	Rechtsrotation
sahf	Store AH into Flags	Register AH im Kennzeichenregister ablegen
sal	Shift Instruction	Linksrotation (arithmetisch)
sar	Shift Instruction	Rechtsrotation (arithmetisch)
sbb	Integer Subtraction with Borrow	Subtraktion mit Übertrag
scas	Compare String Data	Zeichenkettenvergleich
set*cc*	Byte Set on Condition	Setzen eines Bytes unter Berücksichtigung der Bedingung *cc*
sgdt	Store Global Descriptor Table Register	GDT-Register speichern
shl	Shift Instruction	Linksrotation (logisch)
shld	Double Precision Shift Left	Linksrotation (doppelt)
shr	Shift Instruction	Rechtsrotation (logisch)
shrd	Double Precision Shift Right	Rechtsrotation (doppelt)
sidt	Store Interrupt Descriptor Table Register	IDT-Register speichern
sldt	Store Local Descriptor Table Register	LDT-Register speichern

Befehl (Abkürzung)	englische Intel-Spezifikation	deutsche Bedeutung
smsw	Store Machine Status Word	Maschinenstatuswort speichern
stc	Set Carry Flag	Übertragskennzeichen setzen
std	Set Direction Flag	Richtungskennzeichen setzen
sti	Set Interrupt Flag	Unterbrechung-Kennzeichen setzen
stos	Store String Data	Zeichenkette speichern
str	Store Task Register	Taskregister speichern
sub	Integer Subtraction	Subtraktion
test	Logical Compare	Teste Bits
verr	Verify a Segment for Reading	Überprüfung eines Segments zum Lesen
verw	Verify a Segment for Writing	Überprüfung eines Segments zum Schreiben
wait	Wait	Prozessor in Wartezustand versetzen
wbinvd	Write-Back and Invalidate Cache	Cache-Inhalt in Speicher zurückschreiben und Cache löschen
xadd	Exchange and Add	Operandeninhalt austauschen und addieren
xchg	Exchange Register/Memory with Register	Register mit Register/Speicher austauschen
xlat	Table Look-up Translation	Tabellenanpassung
xor	Logical Exclusive OR	Verknüpfung durch exklusives Oder

Tabelle 1. Die Intel-Assembler-Kommandos sind in vielen Fällen als Abkürzung einer längeren Befehlsbeschreibung konzipiert

Die Intel-Kommandos und ihre Funktion

Befehl	Maschinencode	8086	80186	80286	80386	80486	32-Bit-Erweiterung	Funktionsbeschreibung
aaa	37hex	●	●	●	●	●	○	paßt AL nach ASCII-Addition an
aad	D5hex	●	●	●	●	●	○	paßt AL vor ASCII-Division an
aam	D4hex	●	●	●	●	●	○	paßt AL nach ASCII-Multiplikation an
aas	3Fhex	●	●	●	●	●	○	paßt AL nach ASCII-Subtraktion an
adc AL, data8	14hex, data8	●	●	●	●	●	○	addiert data8 zu AL unter Berücksichtigung des Übertragskennzeichens
adc AX, data16	15hex, data16	●	●	●	●	●	●	addiert data16 zu AX unter Berücksichtigung des Übertragskennzeichens
adc r/m16, data16	81hex, mod 010 r/m, data16	●	●	●	●	●	●	addiert data16 zu r/m16 unter Berücksichtigung des Übertragskennzeichens
adc r/m16, data8	83hex, mod 010 r/m, data8	●	●	●	●	●	●	addiert data8 zu r/m16 unter Berücksichtigung des Übertragskennzeichens
adc r/m16, reg16	11hex, mod reg r/m	●	●	●	●	●	●	addiert reg16 zu r/m16 unter Berücksichtigung des Übertragskennzeichens
adc r/m8, data8	80hex, mod 010 r/m, data8	●	●	●	●	●	○	addiert data8 zu r/m8 unter Berücksichtigung des Übertragskennzeichens
adc r/m8, reg8	10hex, mod reg r/m	●	●	●	●	●	○	addiert reg8 zu r/m8 unter Berücksichtigung des Übertragskennzeichens
adc reg16, r/m16	13hex, mod reg r/m	●	●	●	●	●	●	addiert r/m16 zu reg16 unter Berücksichtigung des Übertragskennzeichens
adc reg8, r/m8	12hex, mod reg r/m	●	●	●	●	●	○	addiert r/m8 zu reg8 unter Berücksichtigung des Übertragskennzeichens
add AL, data8	4, data8	●	●	●	●	●	○	addiert data8 zu AL ohne Berücksichtigung des Übertragskennzeichens
add AX, data16	5, data16	●	●	●	●	●	●	addiert data16 zu AX ohne Berücksichtigung des Übertragskennzeichens
add r/m16, data16	81hex, mod 000 r/m, data16	●	●	●	●	●	●	addiert data16 zu r/m16 ohne Berücksichtigung des Übertragskennzeichens
add r/m16, data8	83hex, mod 000 r/m, data8	●	●	●	●	●	●	addiert data8 zu r/m16 ohne Berücksichtigung des Übertragskennzeichens
add r/m16, reg16	1, mod reg r/m	●	●	●	●	●	●	addiert reg16 zu r/m16 ohne Berücksichtigung des Übertragskennzeichens
add r/m8, data8	80hex, mod 000 r/m, data8	●	●	●	●	●	○	addiert data8 zu r/m8 ohne Berücksichtigung des Übertragskennzeichens

Befehl	Maschinencode	8086	80186	80286	80386	80486	32-Bit-Erweiterung	Funktionsbeschreibung
add r/m8, reg8	0, mod reg r/m	●	●	●	●	●	○	addiert reg8 zu r/m8 ohne Berücksichtigung des Übertragskennzeichens
add reg16, r/m16	3, mod reg r/m	●	●	●	●	●	●	addiert r/m16 zu reg16 ohne Berücksichtigung des Übertragskennzeichens
add reg8, r/m8	2, mod reg r/m	●	●	●	●	●	○	addiert r/m8 zu reg8 ohne Berücksichtigung des Übertragskennzeichens
adrsiz:	67hex	○	○	○	●	●	○	bestimmt Adressengröße
and AL, data8	24hex, data8	●	●	●	●	●	○	verknüpft data8 mit AL über die logische Und-Funktion
and AX, data16	25hex, data16	●	●	●	●	●	●	verknüpft data16 mit AX über die logische Und-Funktion
and r/m16, data16	81hex, mod 100 r/m, data16	●	●	●	●	●	●	verknüpft data16 mit r/m16 über die logische Und-Funktion
and r/m16, data8	83hex, mod 100 r/m, data8	●	●	●	●	●	●	verknüpft data8 mit r/m16 über die logische Und-Funktion
and r/m16, reg16	21hex, mod reg r/m	●	●	●	●	●	●	verknüpft reg16 mit r/m16 über die logische Und-Funktion
and r/m8, data8	80hex, mod 100 r/m, data8	●	●	●	●	●	○	verknüpft data8 mit r/m8 über die logische Und-Funktion
and r/m8, reg8	20hex, mod reg r/m	●	●	●	●	●	○	verknüpft reg8 mit r/m8 über die logische Und-Funktion
and reg16, r/m16	23hex, mod reg r/m	●	●	●	●	●	●	verknüpft r/m16 mit reg16 über die logische Und-Funktion
and reg8, r/m8	22hex, mod reg r/m	●	●	●	●	●	○	verknüpft r/m8 mit reg8 über die logische Und-Funktion
arpl r/m16, reg16	63hex, mod reg r/m	○	○	●	●	●	○	korrigiert das RPL-Feld des Selektors r/m16 mit Hilfe von reg16
bound reg16, mem16	62hex, mod reg r/m	○	●	●	●	●	●	überprüft, ob reg16 innerhalb der durch mem16 spezifizierten Grenzen liegt
bsf reg16, r/m16	Fhex, BChex, mod reg r/m	○	○	○	●	●	●	sucht in r/m16 ab Bit 0 das erste gesetzte Bit und speichert seine Position in reg16
bsr reg16, r/m16	Fhex, BDhex, mod reg r/m	○	○	○	●	●	●	sucht in r/m16 rückwärts ab dem höchstwertigen Bit das erste gesetzte Bit und speichert seine Position in reg16
bswap reg32	0Fhex, C8hex mod reg r/m	○	○	○	○	●	○	invertiert reg32, indem es die Reihenfolge der gespeicherten Bytes umkehrt
bt r/m16, data8	0Fhex, BAhex, mod 100 r/m, data8	○	○	○	●	●	●	speichert das durch data8 indizierte Bit von r/m16 im Übertragskennzeichen
bt r/m16, reg16	0Fhex, A3hex, mod reg r/m	○	○	○	●	●	●	speichert das durch reg16 indizierte Bit von r/m16 im Übertragskennzeichen
btc r/m16, data8	0Fhex, BAhex, mod 111 r/m, data8	○	○	○	●	●	●	speichert das durch data8 indizierte Bit von r/m16 im Übertragskennzeichen und komplementiert es
btc r/m16, reg16	0Fhex, BBhex, mod reg r/m	○	○	○	●	●	●	speichert das durch reg16 indizierte Bit von r/m16 im Übertragskennzeichen und komplementiert es
btr r/m16, data8	0Fhex, BAhex, mod 110 r/m, data8	○	○	○	●	●	●	speichert das durch data8 indizierte Bit von r/m16 im Übertragskennzeichen und löscht es
btr r/m16, reg16	0Fhex, B3hex, mod reg r/m	○	○	○	●	●	●	speichert das durch reg16 indizierte Bit von r/m16 im Übertragskennzeichen und löscht es
bts r/m16, data8	0Fhex, BAhex, mod 101 r/m, data8	○	○	○	●	●	●	speichert das durch data8 indizierte Bit von r/m16 im Übertragskennzeichen und setzt es
bts r/m16, reg16	0Fhex, ABhex, mod reg r/m	○	○	○	●	●	●	speichert das durch reg16 indizierte Bit von r/m16 im Übertragskennzeichen und setzt es
call FAR mem	FFhex, mod 011 r/m	●	●	●	●	●	●	ruft ein durch mem angegebenes Far-Unterprogramm auf
call mem16	E8hex mem16	●	●	●	●	●	●	ruft ein durch mem16 spezifiziertes Unterprogramm auf
call offset16	9Ahex, offset16	●	●	●	●	●	●	führt eine durch offset16 näher beschriebene Near-Prozedur aus

Befehl	Maschinencode	8086	80186	80286	80386	80486	32-Bit-Erweiterung	Funktionsbeschreibung
call r/m16	FFhex, mod 010 r/m	●	●	●	●	●	●	führt eine durch r/m16 angegebene Near-Prozedur aus
cbw	98hex	●	●	●	●	●	●	erweitert das Byte im AL-Register um das höchstsignifikante Bit auf ein Wort im AX-Register
cdq	99hex	○	○	○	●	●	●	erweitert Doppelwort in EAX um das höchstsignifikante Bit auf Vierfachwort in EDX:EAX
clc	F8hex	●	●	●	●	●	○	löscht das Übertragskennzeichen
cld	FChex	●	●	●	●	●	○	löscht das Richtungskennzeichen
cli	FAhex	●	●	●	●	●	○	löscht das Unterbrechungskennzeichen
clts	0Fhex, 6	○	○	●	●	●	○	löscht das TS-Bit im CR0-Register
cmc	F5hex	●	●	●	●	●	○	invertiert das Übertragskennzeichen
cmp AL, data8	3Chex, data8	●	●	●	●	●	○	vergleicht data8 mit AL
cmp AX, data16	3Dhex, data16	●	●	●	●	●	●	vergleicht data16 mit AX
cmp r/m16, data16	81hex, mod 111 r/m, data16	●	●	●	●	●	●	vergleicht data16 mit r/m16
cmp r/m16, data8	83hex, mod 111 r/m, data8	●	●	●	●	●	●	vergleicht data8 mit r/m16
cmp r/m16, reg16	39hex, mod reg r/m	●	●	●	●	●	●	vergleicht reg16 mit r/m16
cmp r/m8, data8	80hex, mod 111 r/m, data8	●	●	●	●	●	○	vergleicht data8 mit r/m8
cmp r/m8, reg8	38hex mod reg r/m	●	●	●	●	●	○	vergleicht reg8 mit r/m8
cmp reg16, r/m16	3Bhex, mod reg r/m	●	●	●	●	●	●	vergleicht r/m16 mit reg16
cmp reg8, r/m8	3Ahex, mod reg r/m	●	●	●	●	●	○	vergleicht r/m8 mit reg8
cmpsb	A6hex	●	●	●	●	●	○	vergleicht durch ES:DI indiziertes Byte mit DS:SI
cmpsd	A7hex	○	○	○	●	●	●	vergleicht durch ES:DI indiziertes Doppelwort mit DS:SI
cmpsw	A7hex	●	●	●	●	●	○	vergleicht durch ES:DI indiziertes Wort mit DS:SI
cmpxchg r/m16, reg16	0Fhex, A7hex, mod reg r/m	○	○	○	○	●	●	vergleicht Akkumulator mit r/m16; sind sie gleich, erhält r/m16 den Wert reg16, ansonsten erhält der Akkumulator den Wert r/m16
cmpxchg r/m8, reg8	0Fhex, A6hex, mod reg r/m	○	○	○	○	●	○	vergleicht Akkumulator mit r/m8; sind sie gleich, erhält r/m8 den Wert reg8, ansonsten erhält der Akkumulator den Wert r/m8
cs:	2Ehex	●	●	●	●	●	○	spezifiziert das Codesegment
cwd	99hex	●	●	●	●	●	○	erweitert Wort in AX um das höchstsignifikante Bit auf Doppelwort in DX:AX
cwde	98hex	○	○	○	●	●	●	erweitert das Wort im AX-Register um das höchtsignifikante Bit auf ein Doppelwort im EAX-Register
daa	27hex	●	●	●	●	●	○	paßt AL nach Addition dezimal an
das	2Fhex	●	●	●	●	●	○	paßt AL nach Subtraktion dezimal an
dec AX	48hex	●	●	●	●	●	●	vermindert AX um den Wert 1
dec BP	4Dhex	●	●	●	●	●	●	vermindert BP um den Wert 1
dec BX	4Bhex	●	●	●	●	●	●	vermindert BX um den Wert 1
dec CX	49hex	●	●	●	●	●	●	vermindert CX um den Wert 1
dec DI	4Fhex	●	●	●	●	●	●	vermindert DI um den Wert 1
dec DX	4Ahex	●	●	●	●	●	●	vermindert DX um den Wert 1
dec r/m16	FFhex, mod 001 r/m	●	●	●	●	●	●	vermindert r/m16 um den Wert 1
dec r/m8	FEhex, mod 001 r/m	●	●	●	●	●	○	vermindert r/m8 um den Wert 1
dec SI	4Ehex	●	●	●	●	●	●	vermindert SI um den Wert 1
dec SP	4Chex	●	●	●	●	●	●	vermindert SP um den Wert 1
div AL, r/m8	F6hex, mod 110 r/m	●	●	●	●	●	○	teilt AL durch r/m8 ohne Berücksichtigung des Vorzeichens
div AX, r/m16	F7hex, mod 110 r/m	●	●	●	●	●	●	dividiert AX durch r/m16 ohne Berücksichtigung des Vorzeichens
ds:	3Ehex	●	●	●	●	●	○	spezifiziert das Datensegment
enter data16, data8	C8hex data16, data8	○	●	●	●	●	○	legt Stapelspeicherstruktur für höhere Programmiersprachen an
es:	26hex	●	●	●	●	●	○	spezifiziert das Extrasegment
esc 0	D8hex	●	●	●	●	●	○	legt den folgenden Operanden auf den Datenbus, um ihn von einer externen Einheit lesen zu lassen
esc 1	D9hex	●	●	●	●	●	○	legt den folgenden Operanden auf den Datenbus, um ihn von einer externen Einheit lesen zu lassen

Befehl	Maschinencode	8086	80186	80286	80386	80486	32-Bit-Erweiterung	Funktionsbeschreibung
esc 2	DAhex	●	●	●	●	●	○	legt den folgenden Operanden auf den Datenbus, um ihn von einer externen Einheit lesen zu lassen
esc 3	DBhex	●	●	●	●	●	○	legt den folgenden Operanden auf den Datenbus, um ihn von einer externen Einheit lesen zu lassen
esc 4	DChex	●	●	●	●	●	○	legt den folgenden Operanden auf den Datenbus, um ihn von einer externen Einheit lesen zu lassen
esc 5	DDhex	●	●	●	●	●	○	legt den folgenden Operanden auf den Datenbus, um ihn von einer externen Einheit lesen zu lassen
esc 6	DEhex	●	●	●	●	●	○	legt den folgenden Operanden auf den Datenbus, um ihn von einer externen Einheit lesen zu lassen
esc 7	DFhex	●	●	●	●	●	○	legt den folgenden Operanden auf den Datenbus, um ihn von einer externen Einheit lesen zu lassen
fs:	64hex	○	○	○	●	●	○	spezifiziert das FS-Register
gs:	65hex	○	○	○	●	●	○	spezifiziert das GS-Register
hlt	F4hex	●	●	●	●	●	○	hält den Mikroprozessor an und versetzt ihn in den Wartezustand
idiv AL, r/m8	F6hex, mod 111 r/m	●	●	●	●	●	○	teilt AL durch r/m8 unter Berücksichtigung des Vorzeichens
idiv AX, r/m16	F7hex, mod 111 r/m	●	●	●	●	●	●	dividiert AX durch r/m16 unter Berücksichtigung des Vorzeichens
imul AL, r/m8	F6hex, mod 101 r/m	●	●	●	●	●	○	multipliziert r/m8 mit AL unter Berücksichtigung des Vorzeichens
imul AX, r/m16	F7hex, mod 101 r/m	●	●	●	●	●	●	multipliziert r/m16 mit AX unter Berücksichtigung des Vorzeichens
imul reg16, r/m16	0Fhex, AFhex, mod reg r/m	○	●	●	●	●	●	multipliziert r/m16 mit reg16 unter Berücksichtigung des Vorzeichens
imul reg16, r/m16, data16	69hex, mod reg r/m, data16	○	●	●	●	●	●	multipliziert r/m16 mit reg16 und mit data16 unter Berücksichtigung des Vorzeichens
imul reg16, r/m16, data8	6Bhex, mod reg r/m, data8	○	●	●	●	●	●	multipliziert r/m16 mit reg16 und mit data8 unter Berücksichtigung des Vorzeichens
in AL, data8	E4hex, data8	●	●	●	●	●	○	überträgt einen Wert vom durch data8 spezifizierten Port nach AL
in AL, DX	EChex	●	●	●	●	●	○	überträgt einen Wert vom durch DX spezifizierten Port nach AL
in AX, data8	E5hex, data8	●	●	●	●	●	●	überträgt einen Wert vom durch data8 spezifizierten Port nach AX
in AX, DX	EDhex	●	●	●	●	●	●	überträgt einen Wert vom durch DX spezifizierten Port nach AX
inc AX	40hex	●	●	●	●	●	●	erhöht AX um den Wert 1
inc BP	45hex	●	●	●	●	●	●	erhöht BP um den Wert 1
inc BX	43hex	●	●	●	●	●	●	erhöht BX um den Wert 1
inc CX	41hex	●	●	●	●	●	●	erhöht CX um den Wert 1
inc DI	47hex	●	●	●	●	●	●	erhöht DI um den Wert 1
inc DX	42hex	●	●	●	●	●	●	erhöht DX um den Wert 1
inc r/m16	FFhex, mod 000 r/m	●	●	●	●	●	●	erhöht r/m16 um den Wert 1
inc r/m8	FEhex, mod 000 r/m	●	●	●	●	●	○	erhöht r/m8 um den Wert 1
inc SI	46hex	●	●	●	●	●	●	erhöht SI um den Wert 1
inc SP	44hex	●	●	●	●	●	●	erhöht SP um den Wert 1
insb	6Chex	○	●	●	●	●	○	überträgt Byte vom durch DX adressierten Port nach ES:DI
insd	6Dhex	○	○	○	●	●	●	überträgt Doppelwort vom durch DX adressierten Port nach ES:DI
insw	6Dhex	○	●	●	●	●	○	überträgt Wort vom durch DX adressierten Port nach ES:DI
int3	CChex	●	●	●	●	●	○	führt Software-Unterbrechung 3 aus (Breakpoint)
int data8	CDhex, data8	●	●	●	●	●	○	ruft Software-Unterbrechung data8 aus
into	CEhex	●	●	●	●	●	○	ruft Software-Unterbrechung 4 auf, wenn das Überlaufskennzeichen gesetzt ist
invd	0Fhex, 8	○	○	○	○	●	○	leert den internen Cache und ordnet an, den externen Cache zu löschen
invlpg mem	0Fhex, 1, mod 111 r/m	○	○	○	○	●	○	markiert einen TLB-Eintrag als ungültig

Befehl	Maschinencode	8086	80186	80286	80386	80486	32-Bit-Erweiterung	Funktionsbeschreibung
iret	CFhex	●	●	●	●	●	○	kehrt aus einer Unterbrechung zurück
ja disp16	0Fhex, 87hex, disp16	○	●	●	●	●	●	springt zu disp16, wenn größer
ja disp8	77hex, disp8	●	●	●	●	●	○	springt zu disp8, wenn größer
jae disp16	0Fhex, 83hex, disp16	○	○	○	●	●	●	springt zu disp16, wenn größer oder gleich
jae disp8	73hex, disp8	●	●	●	●	●	○	springt zu disp8, wenn größer oder gleich
jb disp16	0Fhex, 82hex, disp16	○	○	○	●	●	●	springt zu disp16, wenn kleiner
jb disp8	72hex, disp8	●	●	●	●	●	○	springt zu disp8, wenn kleiner
jbe disp16	0Fhex, 86hex, disp16	○	○	○	●	●	●	springt zu disp16, wenn kleiner oder gleich
jbe disp8	76hex, disp8	●	●	●	●	●	○	springt zu disp8, wenn kleiner oder gleich
jcxz disp8	E3hex, disp8	●	●	●	●	●	○	springt zu disp8, wenn CX gleich 0
je disp16	0Fhex, 84hex, disp16	○	○	○	●	●	●	springt zu disp16, wenn gleich
je disp8	74hex, disp8	●	●	●	●	●	○	springt zu disp8, wenn gleich
jg disp16	0Fhex, 8Fhex, disp16	○	○	○	●	●	●	springt zu disp16, wenn größer
jg disp8	7Fhex, disp8	●	●	●	●	●	○	springt zu disp8, wenn größer
jge disp16	0Fhex, 8Dhex, disp16	○	○	○	●	●	●	springt zu disp16, wenn größer oder gleich
jge disp8	7Dhex, disp8	●	●	●	●	●	○	springt zu disp8, wenn größer oder gleich
jl disp16	0Fhex, 8Chex, disp16	○	○	○	●	●	●	springt zu disp16, wenn kleiner
jl disp8	7Chex, disp8	●	●	●	●	●	○	springt zu disp8, wenn kleiner
jle disp16	0Fhex, 8Ehex, disp16	○	○	○	●	●	●	springt zu disp16, wenn kleiner oder gleich
jle disp8	7Ehex, disp8	●	●	●	●	●	○	springt zu disp8, wenn kleiner oder gleich
jmp disp16	E9hex, disp16	●	●	●	●	●	○	springt unbedingt zur Adresse disp16
jmp disp8	EBhex, disp8	●	●	●	●	●	○	springt unbedingt zur Adresse disp8
jmp FAR mem	FFhex, mod 101 r/m	●	●	●	●	●	●	springt unbedingt zur Far-Adresse mem
jmp FAR mem32	EAhex, mem32	●	●	●	●	●	●	springt unbedingt zur Far-Adresse mem32
jmp r/m16	FFhex, mod 100 r/m	●	●	●	●	●	○	springt zu r/m16
jna disp16	0Fhex, 86hex, disp16	○	○	○	●	●	●	springt zu disp16, wenn kleiner oder gleich
jna disp8	76hex, disp8	●	●	●	●	●	○	springt zu disp8, wenn kleiner oder gleich
jnae disp16	0Fhex, 82hex, disp16	○	○	○	●	●	●	springt zu disp16, wenn kleiner
jnae disp8	72hex, disp8	●	●	●	●	●	○	springt zu disp8, wenn kleiner
jnb disp16	0Fhex, 83hex, ,disp16	○	○	○	●	●	●	springt zu disp16, wenn größer oder gleich
jnb disp8	73hex, disp8	●	●	●	●	●	○	springt zu disp8, wenn größer oder gleich
jnbe disp16	0Fhex, 87hex, disp16	○	○	○	●	●	●	springt zu disp16, wenn größer
jnbe disp8	77hex, disp8	●	●	●	●	●	○	springt zu disp8, wenn größer
jne disp16	0Fhex, 85hex, disp16	○	○	○	●	●	●	springt zu disp16, wenn nicht gleich
jne disp8	75hex, disp8	●	●	●	●	●	○	springt zu disp8, wenn nicht gleich
jng disp16	0Fhex, 8Ehex, disp16	○	○	○	●	●	●	springt zu disp16, wenn kleiner oder gleich
jng disp8	7Ehex, disp8	●	●	●	●	●	○	springt zu disp8, wenn kleiner oder gleich
jnge disp16	0Fhex, 8Chex, disp16	○	○	○	●	●	●	springt zu disp16, wenn kleiner
jnge disp8	7Chex, disp8	●	●	●	●	●	○	springt zu disp8, wenn kleiner
jnl disp16	0Fhex, 8Dhex, disp16	○	○	○	●	●	●	springt zu disp16, wenn größer oder gleich
jnl disp8	7Dhex, disp8	●	●	●	●	●	○	springt zu disp8, wenn größer oder gleich
jnle disp16	0Fhex, 8Fhex, disp16	○	○	○	●	●	●	springt zu disp16, wenn größer
jnle disp8	7Fhex, disp8	●	●	●	●	●	○	springt zu disp8, wenn größer
jno disp16	0Fhex, 81hex, disp16	○	○	○	●	●	●	springt zu disp16, wenn kein Überlauf
jno disp8	71hex, disp8	●	●	●	●	●	○	springt zu disp8, wenn kein Überlauf
jnp disp16	0Fhex, 8Bhex, disp16	○	○	○	●	●	●	springt zu disp16, wenn keine Parität
jnp disp8	7Bhex, disp8	●	●	●	●	●	○	springt zu disp8, wenn keine Parität
jns disp16	0Fhex, 89hex, disp16	○	○	○	●	●	●	springt zu disp16, wenn positiv
jns disp8	79hex, disp8	●	●	●	●	●	○	springt zu disp8, wenn positiv
jnz disp16	0Fhex, 85hex, disp16	○	○	○	●	●	●	springt zu disp16, wenn nicht 0
jnz disp8	75hex, disp8	●	●	●	●	●	○	springt zu disp8, wenn nicht 0
jo disp16	0Fhex, 80hex, disp16	○	○	○	●	●	●	springt zu disp16, wenn Überlauf
jo disp8	70hex, disp8	●	●	●	●	●	○	springt zu disp8, wenn Überlauf
jp disp16	0Fhex, 8Ahex, disp16	○	○	○	●	●	●	springt zu disp16, wenn Parität
jp disp8	7Ahex, disp8	●	●	●	●	●	○	springt zu disp8, wenn Parität

Befehl	Maschinencode	8086	80186	80286	80386	80486	32-Bit-Erweiterung	Funktionsbeschreibung
jpe disp16	0Fhex, 8Ahex, disp16	○	○	○	●	●	●	springt zu disp16, wenn Parität gerade
jpe disp8	7Ahex, disp8	●	●	●	●	●	○	springt zu disp8, wenn Parität gerade
jpo disp16	0Fhex, 8Bhex, disp16	○	○	○	●	●	●	springt zu disp16, wenn Parität ungerade
jpo disp8	7Bhex, disp8	●	●	●	●	●	○	springt zu disp8, wenn Parität ungerade
js disp16	0Fhex, 88hex disp16	○	○	○	●	●	●	springt zu disp16, wenn negativ
js disp8	78hex disp8	●	●	●	●	●	○	springt zu disp8, wenn negativ
jz disp16	0Fhex, 84hex, disp16	○	○	○	●	●	●	springt zu disp16, wenn 0
jz disp8	74hex, disp8	●	●	●	●	●	○	springt zu disp8, wenn 0
lahf	9Fhex	●	●	●	●	●	○	kopiert die Kennzeichenregister CF, PF, AF, ZF und SF in die Bits 0, 2, 4, 6 und 7 des AH-Registers
lar reg16, r/m16	Fhex, 2hex, mod reg r/m	○	○	●	●	●	●	überprüft, ob Selektor r/m16 gültig ist und speichert ihn in reg16
lds reg16, mem	C5hex, mod reg r/m	●	●	●	●	●	●	speichert Offset des Zeigers mem in reg16 und Selektor im Segmentregister DS
lea reg16, mem	8Dhex, mod reg r/m	●	●	●	●	●	●	lädt die durch mem festgelegte Adresse in reg16
leave	C9hex	○	●	●	●	●	○	löst die durch »enter« festgelegte Stapel-Speicherstruktur wieder auf
les reg16, mem	C4hex, mod reg r/m	●	●	●	●	●	●	speichert Offset des Zeigers mem in reg16 und Selektor im Segmentregister ES
lfs reg16, mem	0Fhex, B4hex, mod reg r/m	○	○	○	●	●	●	speichert Offset des Zeigers mem in reg16 und Selektor im Segmentregister FS
lgdt mem	0Fhex, 1, mod 010 r/m	○	○	●	●	●	●	lädt das GDT-Register, das die Adresse und den Grenzwert der globalen Deskriptor-Tabelle festlegt
lgs reg16, mem	0Fhex, B5hex, mod reg r/m	○	○	○	●	●	●	speichert Offset des Zeigers mem in reg16 und Selektor im Segmentregister GS
lidt mem	0Fhex, 1, mod 011 r/m	○	○	●	●	●	●	lädt das IDT-Register, das die Adresse und den Grenzwert der Interrupt-Deskriptor-Tabelle festlegt
lldt r/m16	0Fhex, 0, mod 010 r/m	○	○	●	●	●	●	lädt das LDT-Register, das die Local-Descriptor-Tabelle bestimmt
lmsw r/m16	0Fhex, 1, mod 110 r/m	○	○	●	●	●	●	lädt die niederwertigen 16 Bit aus dem CR0-Register, um beispielsweise in den Protected Mode zu schalten
lock	F0hex	●	●	●	●	●	○	sperrt den Bus für die Zeit, die der angegebene Befehl für die Ausführung benötigt
lodsb	AChex	●	●	●	●	●	○	lädt Byte von DS:SI in das AL-Register
lodsd	ADhex	○	○	○	●	●	●	kopiert Doppelwort von DS:SI nach EAX
lodsw	ADhex	●	●	●	●	●	○	lädt Wort von DS:SI nach AX
loop disp8	E2hex, disp8	●	●	●	●	●	○	verringert CX um 1 und springt zu disp8, wenn CX ungleich 0 ist
loope disp8	E1hex, disp8	●	●	●	●	●	○	verringert CX um 1 und springt zu disp8, wenn CX < > 0 und Nullkennzeichen gesetzt ist
loopne disp8	E0hex, disp8	●	●	●	●	●	○	verringert CX um 1 und springt zu disp8, wenn CX < > 0 und Nullkennzeichen nicht gesetzt ist
loopnz disp8	E0hex, disp8	●	●	●	●	●	○	verringert CX um 1 und springt zu disp8, wenn CX < > 0 und Nullkennzeichen nicht gesetzt ist
loopz disp8	E1hex, disp8	●	●	●	●	●	○	verringert CX um 1 und springt zu disp8, wenn CX < > 0 und Nullkennzeichen gesetzt ist
lsl reg16, r/m16	0Fhex, 3, mod reg r/m	○	○	●	●	●	●	lädt Segmentgrenze des zum Selektor r/m16 gehörigen Deskriptors in das Zielregister reg16
lss reg16, mem	0Fhex, B2hex, mod reg r/m	○	●	●	●	●	○	speichert Offset des Zeigers mem in reg16 und Selektor im Segmentregister SS
ltr r/m16	0Fhex, 0, mod 011 r/m	○	○	●	●	●	○	lädt das Taskregister mit dem durch r/m16 festgelegten Selektor
mov AH, data8	B4hex, data8	●	●	●	●	●	○	überträgt data8 nach AH

Befehl	Maschinencode	8086	80186	80286	80386	80486	32-Bit-Erweiterung	Funktionsbeschreibung
mov AL, data8	B0hex, data8	●	●	●	●	●	○	überträgt data8 nach AL
mov AL, [disp]	A0hex, disp	●	●	●	●	●	○	überträgt [disp] nach AL
mov AX, data16	B8hex data16	●	●	●	●	●	●	überträgt data16 nach AX
mov AX, [disp]	A1hex, disp	●	●	●	●	●	●	überträgt [disp] nach AX
mov BH, data8	B7hex, data8	●	●	●	●	●	○	überträgt data8 nach BH
mov BL, data8	B3hex, data8	●	●	●	●	●	○	überträgt data8 nach BL
mov BP, data16	BDhex, data16	●	●	●	●	●	●	überträgt data16 nach BP
mov BX, data16	BBhex, data16	●	●	●	●	●	●	überträgt data16 nach BX
mov CH, data8	B5hex, data8	●	●	●	●	●	○	überträgt data8 nach CH
mov CL, data8	B1hex, data8	●	●	●	●	●	○	überträgt data8 nach CL
mov CRn, reg32	0Fhex, 20, -/n/reg	○	○	○	●	●	○	überträgt reg32 nach CRn
mov CX, data16	B9hex, data16	●	●	●	●	●	●	überträgt data16 nach CX
mov DH, data8	B6hex, data8	●	●	●	●	●	○	überträgt data8 nach DH
mov DI, data16	BFhex, data16	●	●	●	●	●	●	überträgt data16 nach DI
mov DL, data8	B2hex, data8	●	●	●	●	●	○	überträgt data8 nach DL
mov DRn, reg32	0Fhex, 21hex, -/n/reg	○	○	○	●	●	○	überträgt reg32 nach DRn
mov DX, data16	BAhex, data16	●	●	●	●	●	●	überträgt data16 nach DX
mov r/m16, reg16	89hex, mod reg r/m	●	●	●	●	●	●	überträgt reg16 nach r/m16
mov r/m16, sreg	8Chex, mod a/s r/m	●	●	●	●	●	○	überträgt sreg nach r/m16
mov r/m8, reg8	88hex mod reg r/m	●	●	●	●	●	○	überträgt reg8 nach r/m8
mov reg16, data16	C7hex, mod reg r/m, data16	●	●	●	●	●	●	überträgt data16 nach reg16
mov reg16, r/m16	8Bhex, mod reg r/m	●	●	●	●	●	●	überträgt r/m16 nach reg16
mov reg32, CRn	0Fhex, 22hex, -/n/reg	○	○	○	●	●	○	überträgt CRn nach reg32
mov reg32, DRn	0Fhex, 23hex, -/n/reg	○	○	○	●	●	○	überträgt DRn nach reg32
mov reg32, TRn	0Fhex, 26hex, -/n/reg	○	○	○	●	●	○	überträgt TRn nach reg32
mov reg8, data8	C6hex, mod reg r/m, data8	●	●	●	●	●	○	überträgt data8 nach reg8
mov reg8, r/m8	8Ahex, mod reg r/m	●	●	●	●	●	○	überträgt r/m8 nach reg8
mov SI, data16	BEhex, data16	●	●	●	●	●	●	überträgt data16 nach SI
mov SP, data16	BChex, data16	●	●	●	●	●	●	überträgt data16 nach SP
mov sreg, r/m16	8Ehex, mod a/s r/m	●	●	●	●	●	○	überträgt r/m16 nach sreg
mov TRn, reg32	0Fhex, 24hex, -/n/reg	○	○	○	●	●	○	überträgt reg32 nach TRn
mov [disp], AL	A2hex, disp	●	●	●	●	●	○	überträgt AL nach [disp]
mov [disp], AX	A3hex, disp	●	●	●	●	●	●	überträgt AX nach [disp]
movsb	A4hex	●	●	●	●	●	○	überträgt Byte von SI nach ES:DI
movsd	A5hex	○	○	○	●	●	●	überträgt Doppelwort von SI nach ES:DI
movsw	A5hex	●	●	●	●	●	○	überträgt Wort von SI nach ES:DI
movsx reg32, r/m16	0Fhex, BFhex, mod reg r/m	○	○	○	●	●	●	kopiert einen um den Wert des Vorzeichens erweiterten Operanden r/m16 nach reg32
movsx reg32, r/m8	0Fhex, BEhex, mod reg r/m	○	○	○	●	●	●	kopiert einen um den Wert des Vorzeichens erweiterten Operanden r/m8 nach reg32
movzx reg32, r/m16	0Fhex, B7hex, mod reg r/m	○	○	○	●	●	●	kopiert einen mit Nullbits aufgefüllten Operanden r/m16 nach reg32
movzx reg32, r/m8	0Fhex, B6hex, mod reg r/m	○	○	○	●	●	●	kopiert einen mit Nullbits aufgefüllten Operanden r/m8 nach reg32
mul AL, r/m8	F6hex, mod 100 r/m	●	●	●	●	●	○	multipliziert r/m8 mit AL ohne Berücksichtigung des Vorzeichens
mul AX, r/m16	F7hex, mod 100 r/m	●	●	●	●	●	●	multipliziert r/m16 mit AX ohne Berücksichtigung des Vorzeichens
neg r/m16	F7hex, mod 011 r/m	●	●	●	●	●	●	bildet das Zweierkomplement des Operanden r/m16
neg r/m8	F6hex, mod 011 r/m	●	●	●	●	●	○	bildet das Zweierkomplement des Operanden r/m8
nop	90hex	●	●	●	●	●	○	führt keine Operation aus
not r/m16	F7hex, mod 010 r/m	●	●	●	●	●	●	invertiert r/m16 bitweise
not r/m8	F6hex, mod 010 r/m	●	●	●	●	●	○	invertiert den Operanden r/m8 Bit für Bit
opsiz:	66hex	○	○	○	●	●	○	bestimmt Operandengröße
or AL,data8	0Chex, data8	●	●	●	●	●	○	verknüpft data8 mit AL über die logische Oder-Funktion
or AX, data16	0Dhex, data16	●	●	●	●	●	○	verknüpft data16 mit AX über die logische Oder-Funktion
or r/m16, data16	81hex, mod 001 r/m, data16	●	●	●	●	●	●	verknüpft data16 mit r/m16 über die logische Oder-Funktion
or r/m16, data8	83hex, mod 001 r/m, data8	●	●	●	●	●	●	verknüpft data8 mit r/m16 über die logische Oder-Funktion
or r/m16, reg16	9hex, mod reg r/m	●	●	●	●	●	●	verknüpft reg16 mit r/m16 über die logische Oder-Funktion

Befehl	Maschinencode	8086	80186	80286	80386	80486	32-Bit-Erweiterung	Funktionsbeschreibung
or r/m8, data8	80hex, mod 001 r/m, data8	●	●	●	●	●	○	verknüpft data8 mit r/m8 über die logische Oder-Funktion
or r/m8, reg8	8 mod reg r/m	●	●	●	●	●	○	verknüpft reg8 mit r/m8 über die logische Oder-Funktion
or reg16, r/m16	0Bhex, mod reg r/m	●	●	●	●	●	●	verknüpft r/m16 mit reg16 über die logische Oder-Funktion
or reg8, r/m8	0Ahex, mod reg r/m	●	●	●	●	●	○	verknüpft r/m8 mit reg8 über die logische Oder-Funktion
out data8, AL	E6hex, data8	●	●	●	●	●	○	übergibt den in AL gespeicherten Wert an den durch data8 spezifizierten Port
out data8, AX	E7hex, data8	●	●	●	●	●	●	übergibt den in AX gespeicherten Wert an den durch data8 spezifizierten Port
out DX, AL	EEhex	●	●	●	●	●	○	übergibt den in Al gespeicherten Wert an den durch DX spezifizierten Port
out DX, AX	EFhex	●	●	●	●	●	●	übergibt den in AX gespeicherten Wert an den durch DX spezifizierten Port
outsb	6Ehex	●	●	●	●	●	○	übergibt Byte von SI an den durch DX adressierten Port
outsd	6Fhex	○	○	○	●	●	●	übergibt Doppelwort von SI an den durch DX adressierten Port
outsw	6Fhex	●	●	●	●	●	○	übergibt Wort von SI an den durch DX adressierten Port
pop AX	58hex	●	●	●	●	●	●	holt AX vom Stapel
pop BP	5Dhex	●	●	●	●	●	●	holt BP vom Stapel
pop BX	5Bhex	●	●	●	●	●	●	holt BX vom Stapel
pop CS	0Fhex	●	○	○	○	○	○	holt CS vom Stapel
pop CX	59hex	●	●	●	●	●	●	holt CX vom Stapel
pop DI	5Fhex	●	●	●	●	●	●	holt DI vom Stapel
pop DS	1Fhex	●	●	●	●	●	●	holt DS vom Stapel
pop DX	5Ahex	●	●	●	●	●	●	holt DX vom Stapel
pop ES	7	●	●	●	●	●	●	holt ES vom Stapel
pop FS	0Fhex, A1hex	○	○	○	●	●	●	holt FS vom Stapel
pop GS	0Fhex, A9hex	○	○	○	●	●	●	holt GS vom Stapel
pop r/m16	8Fhex, mod reg r/m	●	●	●	●	●	●	holt r/m16 vom Stapel
pop SI	5Ehex	●	●	●	●	●	●	holt SI vom Stapel
pop SP	5Chex	●	●	●	●	●	●	holt SP vom Stapel
pop SS	17hex	●	●	●	●	●	●	holt SS vom Stapel
popa	61hex	○	●	●	●	●	○	holt alle 16-Bit-Register vom Stapel
popad	61hex	○	○	○	●	●	●	holt alle 32-Bit-Register vom Stapel
popf	9Dhex	●	●	●	●	●	○	holt das Kennzeichenregister vom Stapel
popfd	9Dhex	○	○	○	●	●	●	holt das erweiterte Kennzeichenregister vom Stapel
push AX	50hex	●	●	●	●	●	●	legt AX auf dem Stapel ab
push BP	55hex	●	●	●	●	●	●	legt BP auf dem Stapel ab
push BX	53hex	●	●	●	●	●	●	legt BX auf dem Stapel ab
push CS	0Ehex	●	●	●	●	●	●	legt CS auf dem Stapel ab
push CX	51hex	●	●	●	●	●	●	legt CX auf dem Stapel ab
push data16	68hex data16	○	●	●	●	●	○	legt data16 auf dem Stapel ab
push data8	6Ahex, data8	○	●	●	●	●	○	legt data8 auf dem Stapel ab
push DI	57hex	●	●	●	●	●	●	legt DI auf dem Stapel ab
push DS	1Ehex	●	●	●	●	●	●	legt DS auf dem Stapel ab
push DX	52hex	●	●	●	●	●	●	legt DX auf dem Stapel ab
push ES	6	●	●	●	●	●	●	legt ES auf dem Stapel ab
push FS	0Fhex, A0hex	○	○	○	●	●	●	legt FS auf dem Stapel ab
push GS	0Fhex, A8hex	○	○	○	●	●	●	legt GS auf dem Stapel ab
push r/m16	FFhex, mod 110 r/m	●	●	●	●	●	●	legt r/m16 auf dem Stapel ab
push SI	56hex	●	●	●	●	●	●	legt SI auf dem Stapel ab
push SP	54hex	●	●	●	●	●	●	legt SP auf dem Stapel ab
push SS	16hex	●	●	●	●	●	●	legt SS auf dem Stapel ab
pusha	60hex	○	●	●	●	●	○	legt alle 16-Bit-Register auf dem Stapel ab
pushad	60hex	○	○	○	●	●	●	legt alle 32-Bit-Register auf dem Stapel ab
pushf	9Chex	●	●	●	●	●	○	legt das Kennzeichenregister auf dem Stapel ab
pushfd	9Chex	○	○	○	●	●	●	legt das erweiterte Kennzeichenregister auf dem Stapel ab

Befehl	Maschinencode	8086	80186	80286	80386	80486	32-Bit-Erweiterung	Funktionsbeschreibung
rcl r/m16, 1	D1hex, mod 010 r/m	●	●	●	●	●	●	führt einmal eine Bitrotation von r/m16 durch das Übertragskennzeichen nach links durch
rcl r/m16, CL	D3, mod 010 r/m	●	●	●	●	●	●	führt CL mal eine Bitrotation von r/m16 durch das Übertragskennzeichen nach links durch
rcl r/m16w, data8	C1hex, mod 010 r/m, data8	○	○	●	●	●	●	führt data8 mal eine Bitrotation von r/m16 durch das Übertragskennzeichen nach links durch
rcl r/m8, 1hex	D0, mod 010 r/m	●	●	●	●	●	○	führt einmal eine Bitrotation von r/m8 durch das Übertragskennzeichen nach links durch
rcl r/m8, CL	D2hex, mod 010 r/m	●	●	●	●	●	○	führt CL mal eine Bitrotation von r/m8 durch das Übertragskennzeichen nach links durch
rcl r/m8, data8	C0hex, mod 010 r/m, data8	○	○	●	●	●	○	führt data8 mal eine Bitrotation von r/m8 durch das Übertragskennzeichen nach links durch
rcr r/m16, 1	D1hex, mod 011 r/m	●	●	●	●	●	●	führt einmal eine Bitrotation von r/m16 durch das Übertragskennzeichen nach rechts durch
rcr r/m16, CL	D3hex, mod 011 r/m	●	●	●	●	●	●	führt CL mal eine Bitrotation von r/m16 durch das Übertragskennzeichen nach rechts durch
rcr r/m16, data8	C1hex, mod 011 r/m, data8	○	○	●	●	●	●	führt data8 mal eine Bitrotation von r/m16 durch das Übertragskennzeichen nach rechts durch
rcr r/m8, 1	D0hex, mod 011 r/m	●	●	●	●	●	○	führt einmal eine Bitrotation von r/m8 durch das Übertragskennzeichen nach rechts durch
rcr r/m8, CL	D2hex, mod 011 r/m	●	●	●	●	●	○	führt CL mal eine Bitrotation von r/m8 durch das Übertragskennzeichen nach rechts durch
rcr r/m8, data8	C0hex, mod 011 r/m, data8	○	○	●	●	●	○	führt data8 mal eine Bitrotation von r/m8 durch das Übertragskennzeichen nach rechts durch
rep	F3hex	●	●	●	●	●	○	führt den folgenden Zeichenkettenbefehl abhängig vom Zähler CX aus; entspricht repe und repz
repe	F3hex	●	●	●	●	●	○	führt den folgenden Zeichenkettenbefehl abhängig vom Zähler CX aus; handelt es sich dabei um einen cmps- oder scas-Befehl, wird der Befehl nur wiederholt, wenn das Nullkennzeichen gesetzt ist
repne	F2hex	●	●	●	●	●	○	führt den folgenden Zeichenkettenbefehl abhängig vom Zähler CX aus; handelt es sich dabei um einen cmps- oder scas-Befehl, wird der Befehl nur wiederholt, wenn das Nullkennzeichen gelöscht ist
repnz	F2hex	●	●	●	●	●	○	führt den folgenden Zeichenkettenbefehl abhängig vom Zähler CX aus; handelt es sich dabei um einen cmps- oder scas-Befehl, wird der Befehl nur wiederholt, wenn das Nullkennzeichen gelöscht ist
repz	F3hex	●	●	●	●	●	○	führt den folgenden Zeichenkettenbefehl abhängig vom Zähler CX aus; handelt es sich dabei um einen cmps- oder scas-Befehl, wird der Befehl nur wiederholt, wenn das Nullkennzeichen gesetzt ist
ret	C3hex	●	●	●	●	●	○	kehrt zur Adresse zurück, von der aus ein Unterprogramm aufgerufen wurde
ret data16	C2hex, data16	●	●	●	●	●	○	kehrt zur Adresse (plus data16) zurück, von der aus ein Unterprogramm aufgerufen wurde
retf	CBhex	●	●	●	●	●	○	kehrt zur Adresse zurück, von der aus ein Unterprogramm aufgerufen wurde, indem es zusätzlich den Wert des Codesegments vom Stapel liest

Befehl	Maschinencode	8086	80186	80286	80386	80486	32-Bit-Erweiterung	Funktionsbeschreibung
retf data16	CAhex, data16	●	●	●	●	●	○	kehrt zur Adresse (plus data16) zurück, von der ein Unterprogramm aufgerufen wurde, indem es zusätzlich den Wert des Codesegments vom Stapel liest
rol r/m16, 1	D1hex, mod 000 r/m	●	●	●	●	●	●	führt einmal eine Bitrotation von r/m16 nach links durch
rol r/m16, CL	D3hex, mod 000 r/m	●	●	●	●	●	●	führt CL mal eine Bitrotation von r/m16 nach links durch
rol r/m8, 1	D0hex, mod 000 r/m	●	●	●	●	●	○	führt einmal eine Bitrotation von r/m8 nach links durch
rol r/m8, CL	D2hex, mod 000 r/m	●	●	●	●	●	○	führt CL mal eine Bitrotation von r/m8 nach links durch
rol, r/m16, data8	C1hex, mod 000 r/m, data8	○	○	●	●	●	●	führt data8 mal eine Bitrotation von r/m16 nach links durch
rol, r/m8, data8	C0hex, mod 000 r/m, data8	○	○	●	●	●	○	führt data8 mal eine Bitrotation von r/m8 nach links durch
ror r/m16, 1	D1hex, mod 001 r/m	●	●	●	●	●	●	führt einmal eine Bitrotation von r/m16 nach rechts durch
ror r/m16, CL	D3hex, mod 001 r/m	●	●	●	●	●	●	führt CL mal eine Bitrotation von r/m16 nach rechts durch
ror r/m16, data8	C1hex, mod 001 r/m, data8	○	○	●	●	●	●	führt data8 mal eine Bitrotation von r/m16 nach rechts durch
ror r/m8, 1	D0hex, mod 001 r/m	●	●	●	●	●	○	führt einmal eine Bitrotation von r/m8 nach rechts durch
ror r/m8, CL	D2hex, mod 001 r/m	●	●	●	●	●	○	führt CL mal eine Bitrotation von r/m8 nach rechts durch
ror r/m8, data8	C0hex, mod 001 r/m, data8	○	○	●	●	●	○	führt data8 mal eine Bitrotation von r/m8 nach rechts durch
sahf	9Ehex	●	●	●	●	●	○	kopiert das AH-Register in das Kennzeichenregister
sal r/m16, 1	D1hex, mod 100 r/m	●	●	●	●	●	●	führt einmal eine arithmetische Bitrotation von r/m16 nach links durch
sal r/m16, CL	D3hex, mod 100 r/m	●	●	●	●	●	●	führt CL mal eine arithmetische Bitrotation von r/m16 nach links durch
sal r/m16, data8	C1hex, mod 100 r/m, data8	○	○	●	●	●	●	führt data8 mal eine arithmetische Bitrotation von r/m16 nach links durch
sal r/m8, 1	D0hex, mod 100 r/m	●	●	●	●	●	○	führt einmal eine arithmetische Bitrotation von r/m8 nach links durch
sal r/m8, CL	D2hex, mod 100 r/m	●	●	●	●	●	○	führt CL mal eine arithmetische Bitrotation von r/m8 nach links durch
sal r/m8, data8	C0hex, mod 100 r/m, data8	○	○	●	●	●	○	führt data8 mal eine arithmetische Bitrotation von r/m8 nach links durch
sar r/m16, 1	D1hex, mod 111 r/m	●	●	●	●	●	●	führt einmal eine arithmetische Bitrotation von r/m16 nach rechts durch
sar r/m16, CL	D3hex, mod 111 r/m	●	●	●	●	●	●	führt CL mal eine arithmetische Bitrotation von r/m16 nach rechts durch
sar r/m16, data8	C1hex, mod 111 r/m, data8	○	○	●	●	●	●	führt data8 mal eine arithmetische Bitrotation von r/m16 nach rechts durch
sar r/m8, 1	D0hex, mod 111 r/m	●	●	●	●	●	○	führt einmal eine arithmetische Bitrotation von r/m8 nach rechts durch
sar r/m8, CL	D2hex, mod 111 r/m	●	●	●	●	●	○	führt CL mal eine arithmetische Bitrotation von r/m8 nach rechts durch
sar r/m8, data8	C0hex, mod 111 r/m, data8	○	○	●	●	●	○	führt data8 mal eine arithmetische Bitrotation von r/m8 nach rechts durch
sbb AL, data8	1Chex, data8	●	●	●	●	●	○	subtrahiert data8 von AL unter Berücksichtigung des Übertragskennzeichens
sbb AX, data16	1Dhex, data16	●	●	●	●	●	●	subtrahiert data16 von AX unter Berücksichtigung des Übertragskennzeichens
sbb r/m16, data16	81hex, mod 011 r/m, data16	●	●	●	●	●	●	subtrahiert data16 von r/m16 unter Berücksichtigung des Übertragskennzeichens
sbb r/m16, data8	83hex, mod 011 r/m, data8	●	●	●	●	●	●	subtrahiert data8 von r/m16 unter Berücksichtigung des Übertragskennzeichens
sbb r/m16, reg16	19hex, mod reg r/m	●	●	●	●	●	●	subtrahiert reg16 von r/m16 unter Berücksichtigung des Übertragskennzeichens
sbb r/m8, data8	80hex, mod 011 r/m, data8	●	●	●	●	●	○	subtrahiert data8 von r/m8 unter Berücksichtigung des Übertragskennzeichens

Befehl	Maschinencode	8086	80186	80286	80386	80486	32-Bit-Erweiterung	Funktionsbeschreibung
sbb r/m8, reg8	18hex mod reg r/m	●	●	●	●	●	○	subtrahiert reg8 von r/m8 unter Berücksichtigung des Übertragskennzeichens
sbb reg16, r/m16	1Bhex, mod reg r/m	●	●	●	●	●	●	subtrahiert r/m16 von reg16 unter Berücksichtigung des Übertragskennzeichens
sbb reg8, r/m8	1Ahex, mod reg r/m	●	●	●	●	●	○	subtrahiert r/m8 von reg8 unter Berücksichtigung des Übertragskennzeichens
scasb	AEhex	●	●	●	●	●	○	vergleicht Byte in AL mit ES:DI
scasd	AFhex	○	○	○	●	●	●	vergleicht Doppelwort in EAX mit ES:DI
scasw	AFhex	●	●	●	●	●	○	vergleicht Wort in AX mit ES:DI
seta r/m8	0Fhex, 97hex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn darüber
setae r/m8	0Fhex, 93hex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn darüber oder gleich
setb r/m8	0Fhex, 92hex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn darunter
setbe r/m8	0Fhex, 96hex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn kleiner oder gleich
sete r/m8	0Fhex, 94hex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn gleich
setge r/m8	0Fhex, 9Dhex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn größer oder gleich
set r/m8	0Fhex, 9Fhex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1
setl r/m8	0Fhex, 9Chex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn kleiner
setle r/m8	0Fhex, 9Ehex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn kleiner oder gleich
setna r/m8	0Fhex, 96hex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn nicht größer
setnae r/m8	0Fhex, 92hex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn nicht größer oder gleich
setnb r/m8	0Fhex, 93hex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn nicht kleiner
setnbe r/m8	0Fhex, 97hex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn nicht kleiner oder gleich
setne r/m8	0Fhex, 95hex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn nicht gleich
setng r/m8	0Fhex, 9Ehex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn nicht größer
setnge r/m8	0Fhex, 9Chex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn nicht größer oder gleich
setnl r/m8	0Fhex, 9Dhex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn nicht kleiner
setnle r/m8	0Fhex, 9Fhex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn nicht kleiner oder gleich
setno r/m8	0Fhex, 91hex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn kein Überlauf
setnp r/m8	0Fhex, 9Bhex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn keine Parität
setns r/m8	0Fhex, 99hex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn kein Vorzeichen
setnz r/m8	0Fhex, 95hex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn nicht 0
seto r/m8	0Fhex, 90hex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn Überlauf
setp r/m8	0Fhex, 9Ahex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn Parität
setpe r/m8	0Fhex, 9Ahex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn Parität oder gleich
setpo r/m8	0Fhex, 9Bhex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn Parität oder Überlauf
sets r/m8	0Fhex, 98hex mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn Vorzeichen
setz r/m8	0Fhex, 94hex, mod reg r/m	○	○	○	●	●	○	setzt Byte r/m8 auf 1, wenn 0
sgdt mem	0Fhex, 01hex, mod 000 r/m	○	○	●	●	●	○	schreibt das GDT-Register, das die Adresse und den Grenzwert der globalen Deskriptor-Tabelle festlegt, in die Speicheradresse mem

Befehl	Maschinencode	8086	80186	80286	80386	80486	32-Bit-Erweiterung	Funktionsbeschreibung
shl r/m16, 1	D1hex, mod 100 r/m	●	●	●	●	●	●	führt einmal eine logische Bitrotation von r/m16 nach links durch
shl r/m16, CL	D3hex, mod 100 r/m	●	●	●	●	●	●	führt CL mal eine logische Bitrotation von r/m16 nach links durch
shl r/m16, data8	C1hex, mod 100 r/m, data8	○	○	●	●	●	●	führt data8 mal eine logische Bitrotation von r/m16 nach links durch
shl r/m8, 1	D0hex, mod 100 r/m	●	●	●	●	●	○	führt einmal eine logische Bitrotation von r/m8 nach links durch
shl r/m8, CL	D2hex, mod 100 r/m	●	●	●	●	●	○	führt CL mal eine logische Bitrotation von r/m8 nach links durch
shl r/m8, data8	C0hex, mod 100 r/m, data8	○	○	●	●	●	○	führt data8 mal eine logische Bitrotation von r/m8 nach links durch
shld r/m16, reg16, CL	0Fhex, A5hex, mod reg r/m	○	○	○	●	●	●	verknüpft reg16 mit r/m16, verschiebt den resultierenden Wert um CL Bit nach links und speichert den niederwertigen Teil des Ergebnisses wieder in r/m16
shld r/m16, reg16, data8	0Fhex, A4hex, mod reg r/m, data8	○	○	○	●	●	●	verknüpft reg16 mit r/m16, verschiebt den resultierenden Wert um data8 Bit nach links und speichert den niederwertigen Teil des Ergebnisses wieder in r/m16
shr r/m16, 1	D1hex, mod 101 r/m	●	●	●	●	●	●	führt einmal eine logische Bitrotation von r/m16 nach rechts durch
shr r/m16, CL	D3hex, mod 101 r/m	●	●	●	●	●	●	führt CL mal eine logische Bitrotation von r/m16 nach rechts durch
shr r/m16, data8	C1hex, mod 101 r/m, data8	○	○	●	●	●	●	führt data8 mal eine logische Bitrotation von r/m16 nach rechts durch
shr r/m8, 1	D0hex, mod 101 r/m	●	●	●	●	●	○	führt einmal eine logische Bitrotation von r/m8 nach rechts durch
shr r/m8, CL	D2hex, mod 101 r/m	●	●	●	●	●	○	führt CL mal eine logische Bitrotation von r/m8 nach rechts durch
shr r/m8, data8	C0hex, mod 101 r/m, data8	○	○	●	●	●	○	führt data8 mal eine logische Bitrotation von r/m8 nach rechts durch
shrd r/m16, reg16, CL	0Fhex, ADhex, mod reg r/m	○	○	○	●	●	●	verknüpft reg16 mit r/m16, verschiebt den resultierenden Wert um CL Bit nach rechts und speichert den niederwertigen Teil des Ergebnisses wieder in r/m16
shrd r/m16, reg16, data8	0Fhex, AChex, mod reg r/m, data8	○	○	○	●	●	●	verknüpft reg16 mit r/m16, verschiebt den resultierenden Wert um data8 Bit nach rechts und speichert den niederwertigen Teil des Ergebnisses wieder in r/m16
sidt mem	0Fhex, 1, mod 001 r/m	○	○	●	●	●	○	schreibt das IDT-Register, das die Adresse und den Grenzwert der Interrupt-Deskriptor-Tabelle festlegt, in die Speicheradresse mem
sldt r/m16	0Fhex, 0, mod 000 r/m	○	○	●	●	●	○	speichert das LDT-Register, das die Local-Descriptor-Tabelle bestimmt, in r/m16
smsw r/m16	0Fhex, 1, mod 100 r/m	○	○	●	●	●	○	legt das Maschinenstatuswort in r/m16 ab
ss:	36hex	●	●	●	●	●	○	spezifiziert das SS-Register
stc	F9hex	●	●	●	●	●	○	setzt das Übertragskennzeichen
std	FDhex	●	●	●	●	●	○	setzt das Richtungskennzeichen
sti	FBhex	●	●	●	●	●	○	setzt das Interruptkennzeichen
stosb	AAhex	●	●	●	●	●	○	speichert den aktuellen Inhalt von AL in ES:DI
stosd	ABhex	○	○	○	●	●	●	speichert den aktuellen Inhalt von EAX in ES:DI
stosw	ABhex	●	●	●	●	●	○	speichert den aktuellen Inhalt von AX in ES:DI
str r/m16	0Fhex, 0, mod 001 r/m	○	○	○	●	●	○	legt das Taskregister in r/m16 ab
sub AL, data8	2Chex, data8	●	●	●	●	●	○	subtrahiert data8 von AL ohne Berücksichtigung des Übertragskennzeichens
sub AX, data16	2Dhex, data16	●	●	●	●	●	○	subtrahiert data16 von AX ohne Berücksichtigung des Übertragskennzeichens
sub r/m16, data16	81hex, mod 101 r/m, data16	●	●	●	●	●	○	subtrahiert data16 von r/m16 ohne Berücksichtigung des Übertragskennzeichens

Befehl	Maschinencode	8086	80186	80286	80386	80486	32-Bit-Erweiterung	Funktionsbeschreibung
sub r/m16, data8	83hex, mod 101 r/m, data8	●	●	●	●	●	○	subtrahiert data8 von r/m16 ohne Berücksichtigung des Übertragskennzeichens
sub r/m16, reg16	29hex, mod reg r/m	●	●	●	●	●	○	subtrahiert reg16 von r/m16 ohne Berücksichtigung des Übertragskennzeichens
sub r/m8, data8	80hex, mod 101 r/m, data8	●	●	●	●	●	○	subtrahiert data8 von r/m8 ohne Berücksichtigung des Übertragskennzeichens
sub r/m8, reg8	28hex mod reg r/m	●	●	●	●	●	○	subtrahiert reg8 von r/m8 ohne Berücksichtigung des Übertragskennzeichens
sub reg16, r/m16	2Bhex, mod reg r/m	●	●	●	●	●	○	subtrahiert r/m16 von reg16 ohne Berücksichtigung des Übertragskennzeichens
sub reg8, r/m8	2Ahex, mod reg r/m	●	●	●	●	●	○	subtrahiert r/m8 von reg8 ohne Berücksichtigung des Übertragskennzeichens
test AL, data8	A8hex data8	●	●	●	●	●	○	verknüpft data8 mit AL durch die logische Und-Funktion und setzt entsprechend die Kennzeichenregister
test AX, data16	A9hex, data16	●	●	●	●	●	●	verknüpft data16 mit AX durch die logische Und-Funktion und setzt entsprechend die Kennzeichenregister
test r/m16, data16	F7hex, mod 000 r/m, data16	●	●	●	●	●	●	verknüpft data16 mit r/m16 durch die logische Und-Funktion und setzt entsprechend die Kennzeichenregister
test r/m16, reg16	85hex, mod reg r/m	●	●	●	●	●	○	verknüpft reg16 mit r/m16 durch die logische Und-Funktion und setzt entsprechend die Kennzeichenregister
test r/m8, data8	F6hex, mod 000 r/m, data8	●	●	●	●	●	○	verknüpft data8 mit r/m8 durch die logische Und-Funktion und setzt entsprechend die Kennzeichenregister
test r/m8, reg8	84hex, mod reg r/m	●	●	●	●	●	○	verknüpft reg8 mit r/m8 durch die logische Und-Funktion und setzt entsprechend die Kennzeichenregister
verr r/m16	0Fhex, 0, mod 100 r/m	○	○	●	●	●	○	überprüft, ob die aktuelle Prozedur den Selektor r/m16 laden und einen Wert des Speichersegments lesen kann
verw r/m16	0Fhex, 0, mod 101 r/m	○	○	●	●	●	○	überprüft, ob die aktuelle Prozedur den Selektor r/m16 laden und einen Wert in das Speichersegments schreiben kann
wait	9Bhex	●	●	●	●	●	○	versetzt einen Prozessor in den Wartezustand, solange die Busy-Leitung aktiv ist
wbinvd	0Fhex, 9	○	○	○	○	●	○	leert den internen Cache und ordnet an, den Inhalt des externen Cache in den Hauptspeicher zurückzuschreiben und zu löschen
xadd r/m16, reg16	0Fhex, C1hex, mod reg r/m	○	○	○	○	●	○	speichert r/m16 in reg16 und die Summe der ursprünglichen Operanden in r/m16
xadd r/m32, reg32	0Fhex, C2hex, mod reg r/m	○	○	○	○	●	○	speichert r/m32 in reg32 und die Summe der ursprünglichen Operanden in r/m32
xadd r/m8, reg8	0Fhex, C0hex, mod reg r/m	○	○	○	○	●	○	speichert r/m8 in reg8 und die Summe der ursprünglichen Operanden in r/m8
xchg AX, BP	95hex	●	●	●	●	●	●	tauscht den Inhalt von BP und AX aus
xchg AX, BX	93hex	●	●	●	●	●	●	tauscht den Inhalt von BX und AX aus
xchg AX, CX	91hex	●	●	●	●	●	●	tauscht den Inhalt von CX und AX aus
xchg AX, DI	97hex	●	●	●	●	●	●	tauscht den Inhalt von DI und AX aus
xchg AX, DX	92hex	●	●	●	●	●	●	tauscht den Inhalt von DX und AX aus
xchg AX, SI	96hex	●	●	●	●	●	●	tauscht den Inhalt von SI und AX aus
xchg AX, SP	94hex	●	●	●	●	●	●	tauscht den Inhalt von SP und AX aus
xchg r/m16, reg16	87hex, mod reg r/m	●	●	●	●	●	●	tauscht den Inhalt von reg16 und r/m16 aus

Befehl	Maschinencode	8086	80186	80286	80386	80486	32-Bit-Erweiterung	Funktionsbeschreibung
xchg r/m8, reg8	86hex, mod reg r/m	●	●	●	●	●	○	tauscht den Inhalt von reg8 und r/m8 aus
xlat	D7hex	●	●	●	●	●	○	speichert das Byte an der Stelle DS:BX in AL
xor AL, data8	34hex, data8	●	●	●	●	●	○	verknüpft data8 mit AL über die logische Exklusiv-Oder-Funktion
xor AX, data16	35hex, data16	●	●	●	●	●	●	verknüpft data16 mit AX über die logische Exklusiv-Oder-Funktion
xor r/m16, data16	81hex, mod 110 r/m, data16	●	●	●	●	●	●	verknüpft data16 mit r/m16 über die logische Exklusiv-Oder-Funktion
xor r/m16, data8	83hex, mod 110 r/m, data8	●	●	●	●	●	●	verknüpft data8 mit r/m16 über die logische Exklusiv-Oder-Funktion
xor r/m16, reg16	31hex, mod reg r/m	●	●	●	●	●	●	verknüpft reg16 mit r/m16 über die logische Exklusiv-Oder-Funktion
xor r/m8, data8	80hex, mod 110 r/m, data8	●	●	●	●	●	○	verknüpft data8 mit r/m8 über die logische Exklusiv- Oder-Funktion
xor r/m8, reg8	30hex, mod reg r/m	●	●	●	●	●	○	verknüpft reg8 mit r/m8 über die logische Exklusiv-Oder-Funktion
xor reg16, r/m16	33hex, mod reg r/m	●	●	●	●	●	●	verknüpft r/m16 mit reg16 über die logische Exklusiv-Oder-Funktion
xor reg8, r/m8	32hex, mod reg r/m	●	●	●	●	●	○	verknüpft r/m8 mit reg8 über die logische Exklusiv-Oder-Funktion

Tabelle 2. Die Vielfältigkeit des Befehlssatzes der Prozessoren gibt Aufschluß über Ihre Leistungsfähigkeit

Die Intel-Anweisungen in numerischer Reihenfolge

Maschinencode	Befehl	1.Argument	2.Argument	3.Argument	8086	80186	80286	80386	80486	32-Bit-Erweiterung
0, mod reg r/m	add	r/m8,	reg8		●	●	●	●	●	○
1, mod reg r/m	add	r/m16,	reg16		●	●	●	●	●	●
2, mod reg r/m	add	reg8,	r/m8		●	●	●	●	●	○
3, mod reg r/m	add	reg16,	r/m16		●	●	●	●	●	●
4, data8	add	AL,	data8		●	●	●	●	●	○
5, data16	add	AX,	data16		●	●	●	●	●	●
6	push	ES			●	●	●	●	●	●
7	pop	ES			●	●	●	●	●	●
8, mod reg r/m	or	r/m8,	reg8		●	●	●	●	●	○
9, mod reg r/m	or	r/m16,	reg16		●	●	●	●	●	●
0Ahex, mod reg r/m	or	reg8,	r/m8		●	●	●	●	●	○
0Bhex, mod reg r/m	or	reg16,	r/m16		●	●	●	●	●	●
0Chex, data8	or	AL,data8			●	●	●	●	●	○
0Dhex, data16	or	AX,	data16		●	●	●	●	●	○
0Ehex	push	CS			●	●	●	●	●	●
0Fhex	pop	CS			●	○	○	○	○	○
0Fhex, 0hex,mod 000 r/m	sldt	r/m16			○	○	●	●	●	○
0Fhex, 0hex,mod 001 r/m	str	r/m16			○	○	○	●	●	○
0Fhex, 0hex,mod 010 r/m	lldt	r/m16			○	○	●	●	●	●
0Fhex, 0hex,mod 011 r/m	ltr	r/m16			○	○	●	●	●	○
0Fhex, 0hex,mod 100 r/m	verr	r/m16			○	○	●	●	●	○
0Fhex, 0hex,mod 101 r/m	verw	r/m16			○	○	●	●	●	○
0Fhex, 1hex, mod 000 r/m	sgdt	mem			○	○	●	●	●	○
0Fhex, 1hex, mod 001 r/m	sidt	mem			○	○	●	●	●	○
0Fhex, 1hex, mod 010 r/m	lgdt	mem			○	○	●	●	●	●
0Fhex, 1hex, mod 011 r/m	lidt	mem			○	○	●	●	●	●
0Fhex, 1hex, mod 100 r/m	smsw	r/m16			○	○	●	●	●	○
0Fhex, 1hex, mod 110 r/m	lmsw	r/m16			○	○	●	●	●	●
0Fhex, 1hex, mod 111 r/m	invlpg	mem			○	○	○	○	●	○
0Fhex, 2hex, mod reg r/m	lar	reg16,	r/m16		○	○	●	●	●	●
0Fhex, 3hex, mod reg r/m	lsl	reg16,	r/m16		○	○	●	●	●	●
0Fhex, 6hex	clts				○	○	●	●	●	○
0Fhex, 8hex	invd				○	○	○	○	●	○
0Fhex, 9hex	wbinvd				○	○	○	○	●	○
0Fhex, 20hex, -/n/reg	mov	CRn,	reg32		○	○	○	●	●	○
0Fhex, 21hex, -/n/reg	mov	DRn,	reg32		○	○	○	●	●	○
0Fhex, 2, -/n/reg	mov	reg32,	CRn		○	○	○	●	●	○
0Fhex, 23hex, -/n/reg	mov	reg32,	DRn		○	○	○	●	●	○
0Fhex, 24hex, -/n/reg	mov	TRn,	reg32		○	○	○	●	●	○
0Fhex, 26hex, -/n/reg	mov	reg32,	TRn		○	○	○	●	●	○
0Fhex, 80hex, disp16	jo	disp16			○	○	○	●	●	●
0Fhex, 81hex, disp16	jno	disp16			○	○	○	●	●	●

Maschinencode	Befehl	1.Argument	2.Argument	3.Argument	8086	80186	80286	80386	80486	32-Bit-Erweiterung
0Fhex, 8, disp16	jb	disp16			○	○	○	●	●	●
0Fhex, 82hex, disp16	jnae	disp16			○	○	○	●	●	●
0Fhex, 83hex, disp16	jnb	disp16			○	○	○	●	●	●
0Fhex, 83hex, disp16	jae	disp16			○	○	○	●	●	●
0Fhex, 84hex, disp16	je	disp16			○	○	○	●	●	●
0Fhex, 84hex, disp16	jz	disp16			○	○	○	●	●	●
0Fhex, 85hex, disp16	jne	disp16			○	○	○	●	●	●
0Fhex, 85hex, disp16	jnz	disp16			○	○	○	●	●	●
0Fhex, 86hex, disp16	jbe	disp16			○	○	○	●	●	●
0Fhex, 86hex, disp16	jna	disp16			○	○	○	●	●	●
0Fhex, 87hex, disp16	ja	disp16			○	●	●	●	●	●
0Fhex, 87hex, disp16	jnbe	disp16			○	○	○	●	●	●
0Fhex, 88hex, disp16	js	disp16			○	○	○	●	●	●
0Fhex, 89hex, disp16	jns	disp16			○	○	○	●	●	●
0Fhex, 8Ahex, disp16	jp	disp16			○	○	○	●	●	●
0Fhex, 8Ahex, disp16	jpe	disp16			○	○	○	●	●	●
0Fhex, 8Bhex, disp16	jnp	disp16			○	○	○	●	●	●
0Fhex, 8Bhex, disp16	jpo	disp16			○	○	○	●	●	●
0Fhex, 8Chex, disp16	jl	disp16			○	○	○	●	●	●
0Fhex, 8Chex, disp16	jnge	disp16			○	○	○	●	●	●
0Fhex, 8Dhex, disp16	jge	disp16			○	○	○	●	●	●
0Fhex, 8Dhex, disp16	jnl	disp16			○	○	○	●	●	●
0Fhex, 8Ehex, disp16	jle	disp16			○	○	○	●	●	●
0Fhex, 8Ehex, disp16	jng	disp16			○	○	○	●	●	●
0Fhex, 80Fhex, disp16	jg	disp16			○	○	○	●	●	●
0Fhex, 80Fhex, disp16	jnle	disp16			○	○	○	●	●	●
0Fhex, 90hex, mod reg r/m	seto	r/m8			○	○	○	●	●	○
0Fhex, 91hex, mod reg r/m	setno	r/m8			○	○	○	●	●	○
0Fhex, 92hex, mod reg r/m	setb	r/m8			○	○	○	●	●	○
0Fhex, 92hex, mod reg r/m	setnae	r/m8			○	○	○	●	●	○
0Fhex, 93hex, mod reg r/m	setae	r/m8			○	○	○	●	●	○
0Fhex, 93hex, mod reg r/m	setnb	r/m8			○	○	○	●	●	○
0Fhex, 94hex, mod reg r/m	sete	r/m8			○	○	○	●	●	○
0Fhex, 94hex, mod reg r/m	setz	r/m8			○	○	○	●	●	○
0Fhex, 95hex, mod reg r/m	setne	r/m8			○	○	○	●	●	○
0Fhex, 95hex, mod reg r/m	setnz	r/m8			○	○	○	●	●	○
0Fhex, 96hex, mod reg r/m	setbe	r/m8			○	○	○	●	●	○
0Fhex, 96hex, mod reg r/m	setna	r/m8			○	○	○	●	●	○
0Fhex, 97hex, mod reg r/m	seta	r/m8			○	○	○	●	●	○
0Fhex, 97hex, mod reg r/m	setnbe	r/m8			○	○	○	●	●	○
0Fhex, 98hex, mod reg r/m	sets	r/m8			○	○	○	●	●	○
0Fhex, 99hex, mod reg r/m	setns	r/m8			○	○	○	●	●	○
0Fhex, 9Ahex, mod reg r/m	setp	r/m8			○	○	○	●	●	○
0Fhex, 9Ahex, mod reg r/m	setpe	r/m8			○	○	○	●	●	○
0Fhex, 9Bhex, mod reg r/m	setnp	r/m8			○	○	○	●	●	○
0Fhex, 9Bhex, mod reg r/m	setpo	r/m8			○	○	○	●	●	○
0Fhex, 9Chex, mod reg r/m	setl	r/m8			○	○	○	●	●	○
0Fhex, 9Chex, mod reg r/m	setnge	r/m8			○	○	○	●	●	○
0Fhex, 9Dhex, mod reg r/m	setge	r/m8			○	○	○	●	●	○
0Fhex, 9Dhex, mod reg r/m	setnl	r/m8			○	○	○	●	●	○
0Fhex, 9Ehex, mod reg r/m	setle	r/m8			○	○	○	●	●	○
0Fhex, 9Ehex, mod reg r/m	setng	r/m8			○	○	○	●	●	○
0Fhex, 90Fhex, mod reg r/m	setge	r/m8			○	○	○	●	●	○
0Fhex, 90Fhex, mod reg r/m	setnle	r/m8			○	○	○	●	●	○
0Fhex, A0hex	push	FS			○	○	○	●	●	●
0Fhex, A1hex	pop	FS			○	○	○	●	●	●
0Fhex, A3hex, mod reg r/m	bt	r/m16,	reg16		○	○	○	●	●	●
0Fhex, A4hex, mod reg r/m, data8	shldr/m16,	reg16,	data8		○	○	○	●	●	●
0Fhex, A5hex, mod reg r/m	shld	r/m16,	reg16,	CL	○	○	○	●	●	●
0Fhex, A6hex, mod reg r/m	cmpxchg	r/m8,	reg8		○	○	○	○	●	○
0Fhex, A7hex, mod reg r/m	cmpxchg	r/m16,	reg16		○	○	○	○	●	●
0Fhex, A8hex	push	GS			○	○	○	●	●	●
0Fhex, A9hex	pop	GS			○	○	○	●	●	●
0Fhex, ABhex, mod reg r/m	bts	r/m16,	reg16		○	○	○	●	●	●
0Fhex, AChex, mod reg r/m, data8	shrd	r/m16,	reg16,	data8	○	○	○	●	●	●
0Fhex, ADhex, mod reg r/m	shrd	r/m16,	reg16,	CL	○	○	○	●	●	●
0Fhex, A0Fhex, mod reg r/m	imul	reg16,	r/m16		○	●	●	●	●	●
0Fhex, B2hex, mod reg r/m	lss	reg16,	mem		○	●	●	●	●	○
0Fhex, B3hex, mod reg r/m	btr	r/m16,	reg16		○	○	○	●	●	●
0Fhex, B4hex, mod reg r/m	lfs	reg16,	mem		○	○	○	●	●	●
0Fhex, B5hex, mod reg r/m	lgs	reg16,	mem		○	○	○	●	●	●
0Fhex, B6hex, mod reg r/m	movzx	reg32,	r/m8		○	○	○	●	●	●

Maschinencode	Befehl	1.Argument	2.Argument	3.Argument	8086	80186	80286	80386	80486	32-Bit-Erweiterung
0Fhex, B7hex, mod reg r/m	movzx	reg32,	r/m16		○	○	○	●	●	●
0Fhex, BAhex, mod 100 r/m, data8	bt	r/m16,	data8		○	○	○	●	●	●
0Fhex, BAhex, mod 101 r/m, data8	bts	r/m16,	data8		○	○	○	●	●	●
0Fhex, BAhex, mod 110 r/m, data8	btr	r/m16,	data8		○	○	○	●	●	●
0Fhex, BAhex, mod 111 r/m, data8	btc	r/m16,	data8		○	○	○	●	●	●
0Fhex, BBhex, mod reg r/m	btc	r/m16,	reg16		○	○	○	●	●	●
0Fhex, BChex, mod reg r/m	bsf	reg16,	r/m16		○	○	○	●	●	●
0Fhex, BDhex, mod reg r/m	bsr	reg16,	r/m16		○	○	○	●	●	●
0Fhex, BEhex, mod reg r/m	movsx	reg32,	r/m8		○	○	○	●	●	●
0Fhex, B0Fhex, mod reg r/m	movsx	reg32,	r/m16		○	○	○	●	●	●
0Fhex, C0hex, mod reg r/m	xadd	r/m8,	reg8		○	○	○	○	●	○
0Fhex, C1hex, mod reg r/m	xadd	r/m16,	reg16		○	○	○	○	●	○
0Fhex, C2hex, mod reg r/m	xadd	r/m32,	reg32		○	○	○	○	●	○
0Fhex, C8hex, mod reg r/m	bswap	reg32			○	○	○	○	●	○
10hex, mod reg r/m	adc	r/m8,	reg8		●	●	●	●	●	○
11hex, mod reg r/m	adc	r/m16,	reg16		●	●	●	●	●	●
12hex, mod reg r/m	adc	reg8,	r/m8		●	●	●	●	●	○
13hex, mod reg r/m	adc	reg16,	r/m16		●	●	●	●	●	●
14hex, data8	adc	AL,	data8		●	●	●	●	●	○
15hex, data16	adc	AX,	data16		●	●	●	●	●	●
16hex	push	SS			●	●	●	●	●	●
17hex	pop	SS			●	●	●	●	●	●
18hex, mod reg r/m	sbb	r/m8,	reg8		●	●	●	●	●	○
19hex, mod reg r/m	sbb	r/m16,	reg16		●	●	●	●	●	●
1Ahex, mod reg r/m	sbb	reg8,	r/m8		●	●	●	●	●	○
1Bhex, mod reg r/m	sbb	reg16,	r/m16		●	●	●	●	●	●
1Chex, data8	sbb	AL,	data8		●	●	●	●	●	○
1Dhex, data16	sbb	AX,	data16		●	●	●	●	●	●
1Ehex	push	DS			●	●	●	●	●	●
1Fhex	pop	DS			●	●	●	●	●	●
20hex, mod reg r/m	and	r/m8,	reg8		●	●	●	●	●	○
21hex, mod reg r/m	and	r/m16,	reg16		●	●	●	●	●	●
22hex, mod reg r/m	and	reg8,	r/m8		●	●	●	●	●	○
23hex, mod reg r/m	and	reg16,	r/m16		●	●	●	●	●	●
24hex, data8	and	AL,	data8		●	●	●	●	●	○
25hex, data16	and	AX,	data16		●	●	●	●	●	●
26hex	es:				●	●	●	●	●	○
27hex	daa				●	●	●	●	●	○
28hex, mod reg r/m	sub	r/m8,	reg8		●	●	●	●	●	○
29hex, mod reg r/m	sub	r/m16,	reg16		●	●	●	●	●	○
2Ahex, mod reg r/m	sub	reg8,	r/m8		●	●	●	●	●	○
2Bhex, mod reg r/m	sub	reg16,	r/m16		●	●	●	●	●	○
2Chex, data8	sub	AL,	data8		●	●	●	●	●	○
2Dhex, data16	sub	AX,	data16		●	●	●	●	●	○
2Ehex	cs:				●	●	●	●	●	○
2Fhex	das				●	●	●	●	●	○
30hex, mod reg r/m	xor	r/m8,	reg8		●	●	●	●	●	○
31hex, mod reg r/m	xor	r/m16,	reg16		●	●	●	●	●	●
32hex, mod reg r/m	xor	reg8,	r/m8		●	●	●	●	●	○
33hex, mod reg r/m	xor	reg16,	r/m16		●	●	●	●	●	●
34hex, data8	xor	AL,	data8		●	●	●	●	●	○
35hex, data16	xor	AX,	data16		●	●	●	●	●	●
36hex	ss:				●	●	●	●	●	○
37hex	aaa				●	●	●	●	●	○
38hex, mod reg r/m	cmp	r/m8,	reg8		●	●	●	●	●	○
39hex, mod reg r/m	cmp	r/m16,	reg16		●	●	●	●	●	●
3Ahex, mod reg r/m	cmp	reg8,	r/m8		●	●	●	●	●	○
3Bhex, mod reg r/m	cmp	reg16,	r/m16		●	●	●	●	●	●
3Chex, data8	cmp	AL,	data8		●	●	●	●	●	○
3Dhex, data16	cmp	AX,	data16		●	●	●	●	●	●
3Ehex	ds:				●	●	●	●	●	○
3Fhex	aas				●	●	●	●	●	○
40hex	inc	AX			●	●	●	●	●	●
41hex	inc	CX			●	●	●	●	●	●
42hex	inc	DX			●	●	●	●	●	●
43hex	inc	BX			●	●	●	●	●	●
44hex	inc	SP			●	●	●	●	●	●
45hex	inc	BP			●	●	●	●	●	●
46hex	inc	SI			●	●	●	●	●	●

Maschinencode	Befehl	1.Argument	2.Argument	3.Argument	8086	80186	80286	80386	80486	32-Bit-Erweiterung
47hex	inc	DI			●	●	●	●	●	●
48hex	dec	AX			●	●	●	●	●	●
49hex	dec	CX			●	●	●	●	●	●
4Ahex	dec	DX			●	●	●	●	●	●
4Bhex	dec	BX			●	●	●	●	●	●
4Chex	dec	SP			●	●	●	●	●	●
4Dhex	dec	BP			●	●	●	●	●	●
4Ehex	dec	SI			●	●	●	●	●	●
4Fhex	dec	DI			●	●	●	●	●	●
50hex	push	AX			●	●	●	●	●	●
51hex	push	CX			●	●	●	●	●	●
52hex	push	DX			●	●	●	●	●	●
53hex	push	BX			●	●	●	●	●	●
54hex	push	SP			●	●	●	●	●	●
55hex	push	BP			●	●	●	●	●	●
56hex	push	SI			●	●	●	●	●	●
57hex	push	DI			●	●	●	●	●	●
58hex	pop	AX			●	●	●	●	●	●
59hex	pop	CX			●	●	●	●	●	●
5Ahex	pop	DX			●	●	●	●	●	●
5Bhex	pop	BX			●	●	●	●	●	●
5Chex	pop	SP			●	●	●	●	●	●
5Dhex	pop	BP			●	●	●	●	●	●
5Ehex	pop	SI			●	●	●	●	●	●
5Fhex	pop	DI			●	●	●	●	●	●
60hex	pusha				○	●	●	●	●	○
60hex	pushad				○	○	○	●	●	●
61hex	popa				○	●	●	●	●	○
61hex	popad				○	○	○	●	●	●
62hex, mod reg r/m	bound	reg16,	mem16		○	●	●	●	●	●
63hex, mod reg r/m	arpl,	r/m16,	reg16		○	○	●	●	●	○
64hex	fs:				○	○	○	●	●	○
65hex	gs:				○	○	○	●	●	○
66hex	opsiz:				○	○	○	●	●	○
67hex	adrsiz:				○	○	○	●	●	○
68hex, data16	push	data16			○	●	●	●	●	○
69hex, mod reg r/m, data16	imul	reg16,	r/m16,	data16	○	●	●	●	●	●
6Ahex, data8	push	data8			○	●	●	●	●	○
6Bhex, mod reg r/m, data8	imul	reg16,	r/m16,	data8	○	●	●	●	●	●
6Chex	insb				○	●	●	●	●	○
6Dhex	insd				○	○	○	●	●	●
6Dhex	insw				○	●	●	●	●	○
6Ehex	outsb				●	●	●	●	●	○
6Fhex	outsd				○	○	○	●	●	●
6Fhex	outsw				●	●	●	●	●	○
70hex, disp8	jo	disp8			●	●	●	●	●	○
71hex, disp8	jno	disp8			●	●	●	●	●	○
72hex, disp8	jb	disp8			●	●	●	●	●	○
72hex, disp8	jnae	disp8			●	●	●	●	●	○
73hex, disp8	jae	disp8			●	●	●	●	●	○
73hex, disp8	jnb	disp8			●	●	●	●	●	○
74hex, disp8	je	disp8			●	●	●	●	●	○
74hex, disp8	jz	disp8			●	●	●	●	●	○
75hex, disp8	jne	disp8			●	●	●	●	●	○
75hex, disp8	jnz	disp8			●	●	●	●	●	○
76hex, disp8	jbe	disp8			●	●	●	●	●	○
76hex, disp8	jna	disp8			●	●	●	●	●	○
77hex, disp8	ja	disp8			●	●	●	●	●	○
77hex, disp8	jnbe	disp8			●	●	●	●	●	○
78hex, disp8	js	disp8			●	●	●	●	●	○
79hex, disp8	jns	disp8			●	●	●	●	●	○
7Ahex, disp8	jp	disp8			●	●	●	●	●	○
7Ahex, disp8	jpe	disp8			●	●	●	●	●	○
7Bhex, disp8	jnp	disp8			●	●	●	●	●	○
7Bhex, disp8	jpo	disp8			●	●	●	●	●	○
7Chex, disp8	jl	disp8			●	●	●	●	●	○
7Chex, disp8	jnge	disp8			●	●	●	●	●	○
7Dhex, disp8	jge	disp8			●	●	●	●	●	○
7Dhex, disp8	jnl	disp8			●	●	●	●	●	○
7Ehex, disp8	jle	disp8			●	●	●	●	●	○
7Ehex, disp8	jng	disp8			●	●	●	●	●	○
7Fhex, disp8	jg	disp8			●	●	●	●	●	○
7Fhex, disp8	jnle	disp8			●	●	●	●	●	○

Maschinencode	Befehl	1.Argument	2.Argument	3.Argument	8086	80186	80286	80386	80486	32-Bit-Erweiterung
80hex, mod 000 r/m, data8	add	r/m8,	data8		●	●	●	●	●	○
80hex, mod 001 r/m, data8	or	r/m8,	data8		●	●	●	●	●	○
80hex, mod 010 r/m, data8	adc	r/m8,	data8		●	●	●	●	●	○
80hex, mod 011 r/m, data8	sbb	r/m8,	data8		●	●	●	●	●	○
80hex, mod 100 r/m, data8	and	r/m8,	data8		●	●	●	●	●	○
80hex, mod 101 r/m, data8	sub	r/m8,	data8		●	●	●	●	●	○
80hex, mod 110 r/m, data8	xor	r/m8,	data8		●	●	●	●	●	○
80hex, mod 111 r/m, data8	cmp	r/m8,	data8		●	●	●	●	●	○
81hex, mod 000 r/m, data16	add	r/m16,	data16		●	●	●	●	●	●
81hex, mod 001 r/m, data16	or	r/m16,	data16		●	●	●	●	●	●
81hex, mod 010 r/m, data16	adc	r/m16,	data16		●	●	●	●	●	●
81hex, mod 011 r/m, data16	sbb	r/m16,	data16		●	●	●	●	●	●
81hex, mod 100 r/m, data16	and	r/m16,	data16		●	●	●	●	●	●
81hex, mod 101 r/m, data16	sub	r/m16,	data16		●	●	●	●	●	○
81hex, mod 110 r/m, data16	xor	r/m16,	data16		●	●	●	●	●	●
81hex, mod 111 r/m, data16	cmp	r/m16,	data16		●	●	●	●	●	●
83hex, mod 000 r/m, data8	add	r/m16,	data8		●	●	●	●	●	●
83hex, mod 001 r/m, data8	or	r/m16,	data8		●	●	●	●	●	●
83hex, mod 010 r/m, data8	adc	r/m16,	data8		●	●	●	●	●	●
83hex, mod 011 r/m, data8	sbb	r/m16,	data8		●	●	●	●	●	●
83hex, mod 100 r/m, data8	and	r/m16,	data8		●	●	●	●	●	●
83hex, mod 101 r/m, data8	sub	r/m16,	data8		●	●	●	●	●	○
83hex, mod 110 r/m, data8	xor	r/m16,	data8		●	●	●	●	●	●
83hex, mod 111 r/m, data8	cmp	r/m16,	data8		●	●	●	●	●	●
84hex, mod reg r/m	test	r/m8,	reg8		●	●	●	●	●	○
85hex, mod reg r/m	test	r/m16,	reg16		●	●	●	●	●	○
86hex, mod reg r/m	xchg	r/m8,	reg8		●	●	●	●	●	○
87hex, mod reg r/m	xchg	r/m16,	reg16		●	●	●	●	●	●
88hex, mod reg r/m	mov	r/m8,	reg8		●	●	●	●	●	○
89hex, mod reg r/m	mov	r/m16,	reg16		●	●	●	●	●	●
8Ahex, mod reg r/m	mov	reg8,	r/m8		●	●	●	●	●	○
8Bhex, mod reg r/m	mov	reg16,	r/m16		●	●	●	●	●	●
8Chex, mod a/s r/m	mov	r/m16,	sreg		●	●	●	●	●	○
8Dhex, mod reg r/m	lea	reg16,	mem		●	●	●	●	●	●
8Ehex, mod a/s r/m	mov	sreg,	r/m16		●	●	●	●	●	○
8Fhex, mod reg r/m	pop	r/m16			●	●	●	●	●	●
90hex	nop				●	●	●	●	●	○
91hex	xchg	AX,	CX		●	●	●	●	●	●
92hex	xchg	AX,	DX		●	●	●	●	●	●
93hex	xchg	AX,	BX		●	●	●	●	●	●
94hex	xchg	AX,	SP		●	●	●	●	●	●
95hex	xchg	AX,	BP		●	●	●	●	●	●
96hex	xchg	AX,	SI		●	●	●	●	●	●
97hex	xchg	AX,	DI		●	●	●	●	●	●
98hex	cbw				●	●	●	●	●	●
98hex	cwde				○	○	○	●	●	●
99hex	cdq				○	○	○	●	●	●
99hex	cwd				●	●	●	●	●	○
9Ahex, offset16	call	offset16			●	●	●	●	●	●
9Bhex	wait				●	●	●	●	●	○
9Chex	pushf				●	●	●	●	●	○
9Chex	pushfd				○	○	○	●	●	●
9Dhex	popf				●	●	●	●	●	○
9Dhex	popfd				○	○	○	●	●	●
9Ehex	sahf				●	●	●	●	●	○
9Fhex	lahf				●	●	●	●	●	○
A0hex, disp	mov	AL,	[disp]		●	●	●	●	●	○
A1hex, disp	mov	AX,	[disp]		●	●	●	●	●	●
A2hex, disp	mov	[disp],	AL		●	●	●	●	●	○
A3hex, disp	mov	[disp],	AX		●	●	●	●	●	●
A4hex	movsb				●	●	●	●	●	○
A5hex	movsd				○	○	○	●	●	●
A5hex	movsw				●	●	●	●	●	○
A6hex	cmpsb				●	●	●	●	●	○
A7hex	cmpsd				○	○	○	●	●	●
A7hex	cmpsw				●	●	●	●	●	○
A8hex, data8	test	AL,	data8		●	●	●	●	●	○
A9hex, data16	test	AX,	data16		●	●	●	●	●	●
AAhex	stosb				●	●	●	●	●	○
ABhex	stosd				○	○	○	●	●	●
ABhex	stosw				●	●	●	●	●	○
AChex	lodsb				●	●	●	●	●	○
ADhex	lodsd				○	○	○	●	●	●

Maschinencode	Befehl	1.Argument	2.Argument	3.Argument	8086	80186	80286	80386	80486	32-Bit-Erweiterung
ADhex	lodsw				●	●	●	●	●	○
AEhex	scasb				●	●	●	●	●	○
AFhex	scasd				○	○	○	●	●	●
AFhex	scasw				●	●	●	●	●	○
B0hex, data8	mov	AL,	data8		●	●	●	●	●	○
B1hex, data8	mov	CL,	data8		●	●	●	●	●	○
B2hex, data8	mov	DL,	data8		●	●	●	●	●	○
B3hex, data8	mov	BL,	data8		●	●	●	●	●	○
B4hex, data8	mov	AH,	data8		●	●	●	●	●	○
B5hex, data8	mov	CH,	data8		●	●	●	●	●	○
B6hex, data8	mov	DH,	data8		●	●	●	●	●	○
B7hex, data8	mov	BH,	data8		●	●	●	●	●	○
B8hex, data16	mov	AX,	data16		●	●	●	●	●	●
B9hex, data16	mov	CX,	data16		●	●	●	●	●	●
BAhex, data16	mov	DX,	data16		●	●	●	●	●	●
BBhex, data16	mov	BX,	data16		●	●	●	●	●	●
BChex, data16	mov	SP,	data16		●	●	●	●	●	●
BDhex, data16	mov	BP,	data16		●	●	●	●	●	●
BEhex, data16	mov	SI,	data16		●	●	●	●	●	●
BFhex, data16	mov	DI,	data16		●	●	●	●	●	●
C0hex, mod 000 r/m, data8	rol,	r/m8,	data8		○	○	●	●	●	○
C0hex, mod 001 r/m, data8	ror	r/m8,	data8		○	○	●	●	●	○
C0hex, mod 010 r/m, data8	rcl	r/m8,	data8		○	○	●	●	●	○
C0hex, mod 011 r/m, data8	rcr	r/m8,	data8		○	○	●	●	●	○
C0hex, mod 100 r/m, data8	sal	r/m8,	data8		○	○	●	●	●	○
C0hex, mod 100 r/m, data8	shl	r/m8,	data8		○	○	●	●	●	○
C0hex, mod 101 r/m, data8	shr	r/m8,	data8		○	○	●	●	●	○
C0hex, mod 111 r/m, data8	sar	r/m8,	data8		○	○	●	●	●	○
C1hex, mod 000 r/m, data8	rol,	r/m16,	data8		○	○	●	●	●	●
C1hex, mod 001 r/m, data8	ror	r/m16,	data8		○	○	●	●	●	●
C1hex, mod 010 r/m, data8	rcl	r/m16,	data8		○	○	●	●	●	●
C1hex, mod 011 r/m, data8	rcr	r/m16,	data8		○	○	●	●	●	●
C1hex, mod 100 r/m, data8	sal	r/m16,	data8		○	○	●	●	●	●
C1hex, mod 100 r/m, data8	shl	r/m16,	data8		○	○	●	●	●	●
C1hex, mod 101 r/m, data8	shr	r/m16,	data8		○	○	●	●	●	●
C1hex, mod 111 r/m, data8	sar	r/m16,	data8		○	○	●	●	●	●
C2hex, data16	ret	data16			●	●	●	●	●	○
C3hex	ret				●	●	●	●	●	○
C4hex, mod reg r/m	les	reg16,	r/m16		●	●	●	●	●	●
C5hex, mod reg r/m	lds	reg16,	r/m16		●	●	●	●	●	●
C6hex, mod reg r/m, data8	mov	reg8,	data8		●	●	●	●	●	○
C7hex, mod reg r/m, data16	mov	reg16,	data16		●	●	●	●	●	●
C8hex, data16, data8	enter	data16,	data8		○	●	●	●	●	○
C9hex	leave				○	●	●	●	●	○
CAhex, data16	retf	data16			●	●	●	●	●	○
CBhex	retf				●	●	●	●	●	○
CChex	int	3			●	●	●	●	●	○
CDhex, data8	int	data8			●	●	●	●	●	○
CEhex	into				●	●	●	●	●	○
CFhex	iret				●	●	●	●	●	○
D0hex, mod 000 r/m	rol	r/m8,	1		●	●	●	●	●	○
D0hex, mod 001 r/m	ror	r/m8,	1		●	●	●	●	●	○
D0hex, mod 010 r/m	rcl	r/m8,	1		●	●	●	●	●	○
D0hex, mod 011 r/m	rcr	r/m8,	1		●	●	●	●	●	○
D0hex, mod 100 r/m	sal	r/m8,	1		●	●	●	●	●	○
D0hex, mod 100 r/m	shl	r/m8,	1		●	●	●	●	●	○
D0hex, mod 101 r/m	shr	r/m8,	1		●	●	●	●	●	○
D0hex, mod 111 r/m	sar	r/m8,	1		●	●	●	●	●	○
D1hex, mod 000 r/m	rol	r/m16,	1		●	●	●	●	●	●
D1hex, mod 001 r/m	ror	r/m16,	1		●	●	●	●	●	●
D1hex, mod 010 r/m	rcl	r/m16,	1		●	●	●	●	●	●
D1hex, mod 011 r/m	rcr	r/m16,	1		●	●	●	●	●	●
D1hex, mod 100 r/m	sal	r/m16,	1		●	●	●	●	●	●
D1hex, mod 100 r/m	shl	r/m16,	1		●	●	●	●	●	●
D1hex, mod 101 r/m	shr	r/m16,	1		●	●	●	●	●	●
D1hex, mod 111 r/m	sar	r/m16,	1		●	●	●	●	●	●
D2hex, mod 000 r/m	rol	r/m8,	CL		●	●	●	●	●	○
D2hex, mod 001 r/m	ror	r/m8,	CL		●	●	●	●	●	○
D2hex, mod 010 r/m	rcl	r/m8,	CL		●	●	●	●	●	○
D2hex, mod 011 r/m	rcr	r/m8,	CL		●	●	●	●	●	○
D2hex, mod 100 r/m	sal	r/m8,	CL		●	●	●	●	●	○
D2hex, mod 100 r/m	shl	r/m8,	CL		●	●	●	●	●	○
D2hex, mod 101 r/m	shr	r/m8,	CL		●	●	●	●	●	○

Maschinencode	Befehl	1.Argument	2.Argument	3.Argument	8086	80186	80286	80386	80486	32-Bit-Erweiterung
D2hex, mod 111 r/m	sar	r/m8,	CL		●	●	●	●	●	○
D3hex, mod 000 r/m	rol	r/m16,	CL		●	●	●	●	●	●
D3hex, mod 001 r/m	ror	r/m16,	CL		●	●	●	●	●	●
D3hex, mod 010 r/m	rcl	r/m16,	CL		●	●	●	●	●	●
D3hex, mod 011 r/m	rcr	r/m16,	CL		●	●	●	●	●	●
D3hex, mod 100 r/m	sal	r/m16,	CL		●	●	●	●	●	●
D3hex, mod 100 r/m	shl	r/m16,	CL		●	●	●	●	●	●
D3hex, mod 101 r/m	shr	r/m16,	CL		●	●	●	●	●	●
D3hex, mod 111 r/m	sar	r/m16,	CL		●	●	●	●	●	●
D4hex	aam				●	●	●	●	●	○
D5hex	aad				●	●	●	●	●	○
D7hex	xlat				●	●	●	●	●	○
D8hex	esc	0			●	●	●	●	●	○
D9hex	esc	1			●	●	●	●	●	○
DAhex	esc	2			●	●	●	●	●	○
DBhex	esc	3			●	●	●	●	●	○
DChex	esc	4			●	●	●	●	●	○
DDhex	esc	5			●	●	●	●	●	○
DEhex	esc	6			●	●	●	●	●	○
DFhex	esc	7			●	●	●	●	●	○
E0hex, disp8	loopne	disp8			●	●	●	●	●	○
E0hex, disp8	loopnz	disp8			●	●	●	●	●	○
E1hex, disp8	loope	disp8			●	●	●	●	●	○
E1hex, disp8	loopz	disp8			●	●	●	●	●	○
E2hex, disp8	loop	disp8			●	●	●	●	●	○
E3hex, disp8	jcxz	disp8			●	●	●	●	●	○
E4hex, data8	in	AL,	data8		●	●	●	●	●	○
E5hex, data8	in	AX,	data8		●	●	●	●	●	●
E6hex, data8	out	data8,	AL		●	●	●	●	●	○
E7hex, data8	out	data8,	AX		●	●	●	●	●	●
E8hex, mem16	call	mem16			●	●	●	●	●	●
E9hex, disp16	jmp	disp16			●	●	●	●	●	○
EAhex, mem32	jmp	FAR	mem32		●	●	●	●	●	●
EBhex, disp8	jmp	disp8			●	●	●	●	●	○
EChex	in	AL,	DX		●	●	●	●	●	○
EDhex	in	AX,	DX		●	●	●	●	●	●
EEhex	out	DX,	AL		●	●	●	●	●	○
EFhex	out	DX,	AX		●	●	●	●	●	●
F0hex	lock				●	●	●	●	●	○
F2hex	repne				●	●	●	●	●	○
F2hex	repnz				●	●	●	●	●	○
F3hex	rep				●	●	●	●	●	○
F3hex	repe				●	●	●	●	●	○
F3hex	repz				●	●	●	●	●	○
F4hex	hlt				●	●	●	●	●	○
F5hex	cmc				●	●	●	●	●	○
F6hex, mod 000 r/m, data8	test	r/m8,	data8		●	●	●	●	●	○
F6hex, mod 010 r/m	not	r/m8			●	●	●	●	●	○
F6hex, mod 011 r/m	neg	r/m8			●	●	●	●	●	○
F6hex, mod 100 r/m	mul	AL,	r/m8		●	●	●	●	●	○
F6hex, mod 101 r/m	imul	AL,	r/m8		●	●	●	●	●	○
F6hex, mod 110 r/m	div	AL,	r/m8		●	●	●	●	●	○
F6hex, mod 111 r/m	idiv	AL,	r/m8		●	●	●	●	●	○
F7hex, mod 000 r/m, data16	test	r/m16,	data16		●	●	●	●	●	●
F7hex, mod 010 r/m	not	r/m16			●	●	●	●	●	●
F7hex, mod 011 r/m	neg	r/m16			●	●	●	●	●	●
F7hex, mod 100 r/m	mul	AX,	r/m16		●	●	●	●	●	●
F7hex, mod 101 r/m	imul	AX,	r/m16		●	●	●	●	●	●
F7hex, mod 110 r/m	div	AX,	r/m16		●	●	●	●	●	●
F7hex, mod 111 r/m	idiv	AX,	r/m16		●	●	●	●	●	●
F8hex	clc				●	●	●	●	●	○
F9hex	stc				●	●	●	●	●	○
FAhex	cli				●	●	●	●	●	○
FBhex	sti				●	●	●	●	●	○
FChex	cld				●	●	●	●	●	○
FDhex	std				●	●	●	●	●	○
FEhex, mod 000 r/m	inc	r/m8			●	●	●	●	●	○
FEhex, mod 001 r/m	dec	r/m8			●	●	●	●	●	○
FFhex, mod 000 r/m	inc	r/m16			●	●	●	●	●	●
FFhex, mod 001 r/m	dec	r/m16			●	●	●	●	●	●
FFhex, mod 010 r/m	call	r/m16			●	●	●	●	●	●
FFhex, mod 011 r/m	call	FAR	mem		●	●	●	●	●	●
FFhex, mod 100 r/m	jmp	r/m16			●	●	●	●	●	○

Maschinencode	Befehl	1.Argument	2.Argument	3.Argument	8086	80186	80286	80386	80486	32-Bit-Erweiterung
FFhex, mod 101 r/m	jmp	FAR	mem		•	•	•	•	•	•
FFhex, mod 110 r/m	push	r/m16			•	•	•	•	•	•

Tabelle 3. Eine numerisch sortierte Tabelle erleichtert den direkten Umgang mit Maschinencode

Der Parameter »reg« bestimmt ein Register

reg	Register
000	AX
001	CX
010	DX
011	BX
100	SP
101	BP
110	SI
111	DI

Tabelle 4. Der Reg-Parameter bestimmt eines der acht 16-Bit-Register

Die Kombination von »r/m« und »mod« definiert eine Adressierungsart

r/m	mod=00	mod=01	mod=10	mod=11
000	BX+SI	BX+SI+disp8	BX+SI+disp16	AL oder AX
001	BX			
+DI	BX+DI+disp8	BX+DI+disp16	CL oder CX	
010	BP+SI	BP+SI+disp8	BP+SI+disp16	DL oder DX
011	BP+DI	BP+DI+disp8	BP+DI+disp16	BL oder BX
100	SI	SI+disp8	SI+disp16	AH oder SP
101	DI	DI+disp8	DI+disp16	CH oder BP
110	addr	BP+disp8	BP+disp16	DH oder SI
111	BX	BX+disp8	BX+disp16	BH oder DI

Tabelle 5. Die Kombinationen der einzelnen Parameter legt die Adressierungsart fest

Die Standardparameter der Assembler-Befehle

addr	Adresse einer Speicherzelle
data8	8-Bit-Datenwert
data16	16-Bit-Datenwert
disp8	8-Bit-Adresse, die zu der dem aktuellen Befehl folgenden Adresse hinzugezählt wird
disp16	16-Bit-Adresse, die zu der dem aktuellen Befehl folgenden Adresse hinzugezählt wird
mem16	Inhalt der adressierten Speicherzelle
offset16	Distanzangabe

Tabelle 6. Die Aufstellung der in den Tabellen 2 und 3 verwendeten Parameter erleichtert das Verständnis

Änderungen der Parameter bei einer 32-Bit-Erweiterung

Standard	32-Bit-Erweiterung	Standard	32-Bit-Erweiterung
AX	EAX	mem16	mem32
BP	EBP	mem32	mem48
BX	EBX	offset16	offset32
CX	ECX	r/m16	r/m32
data16	data32	reg16	reg32
DI	EDI	SI	ESI
DX	EDX	SP	ESP

Tabelle 7. Ab dem 80386er ändert sich durch die 32-Bit-Erweiterung die Operandengröße

Utilities gesucht

Programme, die den Bildschirminhalt auf den Kopf stellen, suchen unsere Leser sicherlich nicht. Aber einen wissenschaftlichen Taschenrechner, ein Festplatten-Backup-Programm, einen Extended-Memory-Treiber, Druckerspooler oder Cache-Speicher. Zögern Sie deshalb nicht, schicken Sie uns unverbindlich die aktuelle Version Ihres Programms und einen Ausdruck der Dokumentation.
Falls Sie sich vorweg informieren möchten: Zwischen 15.00 und 17.00 Uhr stehen wir Ihnen an allen Werktagen unter der Rufnummer (0 81 21) 7 69-0 zur Verfügung.

Schicken Sie Ihre Unterlagen an folgende Adresse:

Redaktion DOS International, »DOS Software«,
Gruberstraße 46c, 8011 Poing

Treiberprogrammierung

Flatterhafte Config-Datei

In der »config.sys« geben Sie die Treiber an, die Sie während des Bootvorgangs laden möchten. Ist der Rechner erst einmal eingeschaltet und läuft der Bootvorgang, können Sie keinen Einfluß mehr auf die Daten in der »config.sys« nehmen. Oder doch? Daß auch hier das letzte Wort noch nicht gesprochen ist, zeigt Ihnen dieser Beitrag.

Situationsbeschreibung: Herr X ist begeisterter PC-Anwender und stolz auf sein schier unerschöpfliches Sammelsurium an Software, das er sich im Laufe der Jahre angelegt hat. Da findet man das Feinste vom Feinen in Sachen Text- und Datenverarbeitung, Kalkulation und Grafik. Doch leider bevorzugt jedes seiner Lieblingsprogramme ganz spezielle Treiber, RAM-Disk-Größen, mal Expansionsspeicher, mal Erweiterungsspeicher und so weiter. Bisher blieb ihm nichts anderes übrig, als sich für jede Anwendung eine spezielle Config-Datei zu schreiben, die er vor dem Laden des zugehörigen Programms durch Umbenennen zur aktuellen »config.sys« erklärte und anschließend den Resetknopf zu drücken.

Doch Hilfe ist für Herrn X in Sicht. Mit »cfgedit.sys« (Listing 1) ändert er den Inhalt der »config.sys« während des Bootvorgangs, ohne die Config-Datei selbst anzutasten. Wie das aussieht, zeigt das Bild.

Bewerkstelligen läßt sich der Editiervorgang natürlich nur mit einem Device-Treiber, der sich während der Abarbeitung der »config.sys« in den Bootvorgang einklinkt. Cfgedit nehmen Sie über

```
device=cfgedit.sys
```

in die »config.sys« auf. MS-DOS ruft Cfgedit während des Bootens auf, woraufhin der Treiber zirka zwei Sekunden lang auf einen Tastendruck wartet. Drücken Sie nun die Escape-Taste oder gar keine Taste, bearbeitet MS-DOS den nächsten Befehl aus der »config.sys«. Ansonsten stellt Cfgedit die nachfolgenden (maximal 20) Zeilen zum Bearbeiten zur Verfügung. Als Endemarkierung für den zu bearbeitenden Bereich geben Sie in der »config.sys« die Zeile

```
device=endedit.sys
```

ein. Falls Cfgedit diesen Eintrag nach 20 Zeilen nicht findet, bricht es mit einer entsprechenden Fehlermeldung ab. Übrigens führt diese Markierung bei der Abarbeitung der »config.sys« zu einer Fehlermeldung, die Sie jedoch getrost ignorieren können.

Cfgedit kann Zeilen nur im Rahmen ihrer bereits bestehenden Länge editieren; Sie können einen Eintrag also nicht verlängern. Dieses Manko läßt sich aber leicht beheben, indem Sie bereits in der »config.sys« am Ende der entsprechenden Zeilen ein paar Leerzeichen anhängen.

Beim Bootvorgang liest MS-DOS den Inhalt der »config.sys«, kürzt die einzelnen Einträge ab und legt den Inhalt an einer Speicheradresse ab. Allerdings kürzt MS-DOS nur die Befehlswörter ab. So wird aus

```
device=treiber.sys
```

die Kurzform »DTREIBER.SYS«, mit der Abkürzung »D« für »device«, oder aus

```
files=25
```

entsteht »F25«. Die Abkürzungen und ihre Bedeutung sind in Tabelle 1 aufgelistet. Jeder Eintrag ist mit einem »Carriage-Return« (ASCII-Wert 13) und »Linefeed« (ASCII-Wert 10) abgeschlossen.

Anschließend führt MS-DOS die einzelnen Befehle aus. Trifft es dabei auf ein »D«, also einen Device-Befehl, lädt das Betriebssystem den Treiber in den Speicher und ruft die Routine auf, die bei der Adresse in den Bytes 6 und 7 (»dw strat)« des Treiberkopfes (Tabelle 2) angegeben sein muß: die »Strategieroutine«. Das Registerpaar »ES:BX« zeigt dann auf den sogenannten »Anforderungsblock« des Treibers, der eine Fülle von Informationen enthält. Damit diese Adresse nicht verlorengeht, speichert die Strategieroutine »Strat« sie in »Save_es« und »Save_bx«.

Abkürzung	Bedeutung
D	DEVICE=
F	FILES=
B	BUFFERS=
L	LASTDRIVE=
C	BREAK=
Q	COUNTRY=
P	DRIVPARM=
X	FCBS=
S	SHELL=
K	STACKS=

Tabelle 1. So kürzt MS-DOS die Befehle aus der »config.sys« ab

Interrupt- und Strategieroutine: die Schnittstelle zu MS-DOS

Als nächstes wird die Interruptroutine aufgerufen, deren Offset in den Bytes 8 und 9 im Treiberkopf verzeichnet sein muß (»dw Intr«). Bei Cfgedit heißt sie »Intr« und ist zugleich das Hauptprogramm, das Unterprogramme aufruft und mit dem Betriebssystem kommuniziert.

Nach der Sicherung wichtiger Register und der Flags lädt »Intr« die Register »ES« und »BX« mit den Werten aus den beiden Variablen »Save_es« beziehungsweise »Save_bx«, so daß das Registerpaar wieder auf den Anforderungsblock zeigt. Der Aufruf der Prozedur »Init« gibt die Installationsmeldung von Cfgedit auf den Bildschirm aus. Anschließend wartet »Init« zirka zwei Sekunden auf einen Tastendruck. (Falls das bei Ihnen zu einem Reaktionstest ausarten sollte, können Sie die Wartezeit durch Erhöhen der Konstanten »Delay« heraufsetzen.) Ein kurzer Druck auf die Escape-Taste beendet Cfgedit, bevor die Wartezeit verstrichen ist. Jede andere Taste startet den Editiervorgang.

Den ersten Schritt macht die Routine »ListConfig«. Sie bekommt von »Intr« in »ES:DI« einen Zeiger mit auf den Weg, der auf den oben erwähnten Speicherbereich der abgekürzten Befehle aus der »config.sys« zeigt. Dieser Zeiger steht in den Bytes 18 bis 21 des Anforderungsblocks, auf den »ES:BX« zeigt. Das Registerpaar »ES:DI« zeigt nun aber nicht auf den Anfang dieses Speicherbereichs, sondern auf das Zeichen nach dem Device-Befehl von Cfgedit, also auf das »c« von »cfgedit.sys«. Dieser Eintrag interessiert jedoch nicht, da ja erst die nachfolgenden Befehle der »config.sys« bearbeitet werden sollen. Also sucht die Routine »ListConfig« durch Erhöhen des im Registerpaar »ES:DI« stehenden Zeigers zuerst nach dem abschließenden »LineFeed« und übergeht die entsprechenden Zeichen (siehe Label »Next_Ch«).

Ab »LoopL« wird es ernst. »ListConfig« holt die Zeichen einzeln aus dem Speicherbereich, prüft bei einem gefundenen »Linefeed« über die Prozedur »CheckLast« auf die Endemarkierung »endedit.sys« und trägt jedes Zeichen in die Variable »Content« ein. »Content« ist eine Komponente der Struktur »Line«. Am Ende von »ListConfig« steht jeder Eintrag samt seiner errechneten Länge (»Len«) in einer separaten Struktur, in der er weiterverarbeitet wird.

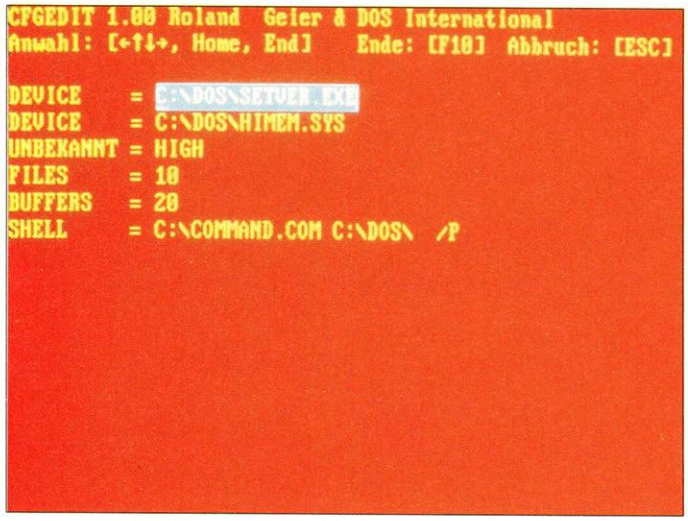
Findet »CheckLast« auch nach 20 Einträgen die Endemarkierung noch nicht, gibt die Routine einen Piepton als Fehlermeldung aus, und Cfgedit wird beendet. Andernfalls schreibt »ListConfig« die Einträge ab dem Label »EndList« auf den Bildschirm, wozu es sich der Ausgaberoutine »OutLine« bedient. Diese wandelt die Befehlsabkürzungen aus Tabelle 1 zudem in die zugehörigen Zeichenketten zurück. Der erste Eintrag steht dann, invers markiert, zum Editieren bereit (Bild).

Das Editieren übernimmt die Routine »Handler«, die über die Prozedur »GetKey« den Tastencode abfragt. Je nach Taste liefert »GetKey« in den Registern »CL«, »AL« und dem Carry-Flag die benötigten Informationen zurück.

Handelt es sich bei der Taste um eine »Steuertaste« (zum Beispiel [Cursor links], [Cursor rechts], [Cursor oben], [Cursor unten], [Ende] und [Pos1]), springt »Handler« über die Tabelle »Jmp_Tab« zur dafür zuständigen Routine.

Falls Sie einen Eintrag mit [Cursor oben] oder [Cursor unten] neu anwählen, übernehmen die entsprechenden Routinen (»KeyUp«, »KeyDn«) auch die Berechnung der zugehörigen neuen Strukturadresse. »Normale« Tasten bewirken in »Handler« die Änderung des entsprechenden Zeichens in der zugeordneten Struktur.

Die Taste [Esc] bricht das Bearbeiten und damit Cfgedit jederzeit mit einer entsprechenden Meldung ab, [F10] führt zur Übernahme aller Änderungen. Die Routine »Move« schreibt den Inhalt (»Content«) der Strukturvariablen jeder bearbeiteten Zeile an die entsprechende Stelle des Speichers zurück. MS-DOS behandelt diese neuen Zeilen so, als wären es die ursprünglichen Zeilen aus der geladenen »config.sys«.



»cfgedit.sys« stellt die zu verändernden Einträge der »config.sys« in einem Editor bereit

Vorher jedoch muß Cfgedit dem Betriebssystem mitteilen, an welche Speicherstelle MS-DOS nach Beenden von Cfgedit den nächsten Treiber laden kann. Da Cfgedit seine Aufgabe bereits erfüllt hat, kann MS-DOS getrost den Speicher von Cfgedit selbst verwenden, es genügt daher die Angabe der ersten belegten Speicherstelle im Codesegment (Label »Start«). Im Anforderungsblock ist ab Offset 14 Platz für diese Information.

Wann Cfgedit fertig ist, erfährt MS-DOS über das Statuswort ab Offset 3 des Anforderungsblocks. Der Wert 0100hex ist der Code für »Aufgabe erledigt, ich bin fertig«.

Um den Treiber zu verwenden, assemblieren Sie »cfgedit.asm« (Listing 2) mit

```
tasm cfgedit
```

oder

```
masm cfgedit
```

und linken die Obj-Datei mit

```
tlink cfgedit
```

oder

```
link cfgedit
```

Anschließend wandeln Sie die Exe-Datei mit

```
exe2bin cfgedit cfgedit.sys
```

in die benötigte Form um. Wenn Sie den Turbo-Linker verwenden, können Sie sich den Aufruf von Exe2bin sparen, wenn Sie beim Linken den Parameter »/t« angeben. Der Linker erzeugt dann automatisch eine Com-Datei, die Sie durch Eingabe von

```
ren cfgedit.com cfgedit.sys
```

umbenennen. Benutzen Sie den MASM, so muß dieser mindestens die Versionsnummer 5.0 tragen, da er sonst einige Pseudoopcodes nicht kennt.

Wenn Sie beim Ändern der Einträge der »config.sys« einen Fehler machen, und das Betriebssystem dadurch einen Treiber nicht finden kann, ist das nicht tragisch. Die damit verbundene Fehlermeldung zieht keine Folgen nach sich

Cfgedit arbeitet problemlos mit MS-DOS zusammen; Schwierigkeiten gibt es hingegen mit DR-DOS. Dort bricht der Treiber mit der Meldung »Ich kann die Endemarkierung »ENDEDIT.SYS« nicht finden« ab.

Herrn X sei mit Cfgedit ein freundlicher Gruß entboten.

(Roland Geier/rm)

Offset	Bedeutung
0	Zeiger auf den nächsten Treiber
4	Geräte-Attribut
6	Offsetadresse der Strategieroutine
8	Offsetadresse der Interruptroutine
0Ahex	Der Name des Treibers bei Zeichentreibern oder die Anzahl der verwalteten Geräte bei Blocktreibern (der Eintrag ist 8 Byte groß)

Tabelle 2. Der Kopf eines Gerätetreibers

```
 1: FF FF FF FF ØØ 8Ø 4F Ø6 5A Ø6 43 4F 4E 58 58 58
 2: 58 24 ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØA ØD 24
 3: 44 45 56 49 43 45 2Ø 2Ø 2Ø 2Ø 3D 2Ø 24 46 49 4C
 4: 45 53 2Ø 2Ø 2Ø 2Ø 2Ø 3D 2Ø 24 42 55 46 46 45 52
 5: 53 2Ø 2Ø 2Ø 3D 2Ø 24 4C 41 53 54 44 52 49 56 45
 6: 2Ø 3D 2Ø 24 42 52 45 41 4B 2Ø 2Ø 2Ø 2Ø 2Ø 3D 2Ø
 7: 24 43 4F 55 4E 54 52 59 2Ø 2Ø 2Ø 3D 2Ø 24 44 52
 8: 49 56 5Ø 41 52 4D 2Ø 2Ø 3D 2Ø 24 46 43 42 53 2Ø
 9: 2Ø 2Ø 2Ø 2Ø 2Ø 3D 2Ø 24 53 48 45 4C 4C 2Ø 2Ø 2Ø
1Ø: 2Ø 2Ø 3D 2Ø 24 53 54 41 43 4B 53 2Ø 2Ø 2Ø 2Ø 3D
11: 2Ø 24 55 4E 42 45 4B 41 4E 4E 54 2Ø 3D 2Ø 24 63
12: ØB 87 ØB FE ØA 98 ØA AC ØB C1 ØB 2Ø 2Ø 2Ø 2Ø 2Ø
13: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
14: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
15: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
16: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ 2Ø
17: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
18: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
```

```
 19: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 2Ø: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 21: 2Ø 2Ø ØØ 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 22: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 23: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 24: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 25: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 26: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 27: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 28: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 29: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ 2Ø 2Ø 2Ø 2Ø 2Ø
 3Ø: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 31: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 32: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 33: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ 2Ø
 34: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 35: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 36: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 37: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 38: 2Ø 2Ø ØØ 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 39: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 4Ø: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 41: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 42: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 43: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 44: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 45: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 46: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ 2Ø 2Ø 2Ø 2Ø 2Ø
 47: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 48: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 49: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 5Ø: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ 2Ø
 51: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 52: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 53: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 54: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 55: 2Ø 2Ø ØØ 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 56: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 57: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 58: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 59: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 6Ø: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 61: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 62: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 63: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ 2Ø 2Ø 2Ø 2Ø 2Ø
 64: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 65: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 66: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 67: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ 2Ø
 68: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 69: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 7Ø: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 71: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 72: 2Ø 2Ø ØØ 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 73: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 74: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 75: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 76: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 77: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 78: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 79: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 8Ø: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ 2Ø 2Ø 2Ø 2Ø 2Ø
 81: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 82: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 83: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 84: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ 2Ø
 85: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 86: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 87: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 88: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 89: 2Ø 2Ø ØØ 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 9Ø: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 91: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 92: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 93: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 94: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 95: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 96: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 97: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ 2Ø 2Ø 2Ø 2Ø
 98: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
 99: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
1ØØ: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
1Ø1: 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ 2E
1Ø2: 89 1E 12 ØØ 2E 8C Ø6 14 ØØ CB 5Ø 53 51 52 1E 9C
1Ø3: ØE 1F 8E Ø6 14 ØØ 8B 1E 12 ØØ E8 7Ø ØØ 72 31 26
1Ø4: 8B 7F 12 26 8B 47 14 8E CØ A3 BF Ø6 E8 86 Ø1 72
1Ø5: 1F E8 C7 Ø3 73 17 53 32 FF B2 ØØ B6 18 B4 Ø2 57
1Ø6: CD 1Ø 5F 5B BA C1 Ø6 B4 Ø9 CD 21 EB Ø3 E8 B1 Ø2
1Ø7: 8E Ø6 14 ØØ 8B 1E 12 ØØ 26 C7 47 ØE ØØ ØØ 26 8C
1Ø8: 4F 1Ø 26 81 4F Ø3 ØØ Ø1 9D 1F 5A 59 5B 58 CB ØØ
1Ø9: ØØ 43 46 47 45 44 49 54 2Ø 61 62 67 65 62 72 6F
11Ø: 63 68 65 6E 2E 2E 2E ØØ ØA ØD ØA ØD 24 Ø6 53 C6
111: Ø6 16 ØØ Ø4 BA 1D ØØ B4 Ø9 CD 21 BA 64 Ø7 CD 21
112: BA 96 Ø7 CD 21 B9 AØ ØF B4 ØB CD 21 3C FF 75 57
113: B4 Ø7 CD 21 3C 1B 74 4C 53 32 FF B2 ØØ B6 ØØ B4
114: Ø2 57 CD 1Ø 5F 5B B9 DØ Ø7 32 FF B3 4E BØ 2Ø B4
115: Ø9 57 CD 1Ø 5F BA 64 Ø7 B4 Ø9 CD 21 BA C7 Ø7 CD
116: 21 B9 Ø1 ØØ BØ 1B B4 Ø9 57 CD 1Ø 5F B4 Ø3 57 CD
117: 1Ø 5F FE C2 B4 Ø2 57 CD 1Ø 5F BA D1 Ø7 B4 Ø9 CD
118: 21 F8 EB ØD F9 EB ØA 83 F9 Ø1 75 Ø3 F9 EB Ø2 E2
119: 97 5B Ø7 C3 43 46 47 45 44 49 54 2Ø 31 2E 3Ø 3Ø
12Ø: 2Ø 52 6F 6C 61 6E 64 ØØ 2Ø 47 65 69 65 72 2Ø 26
121: 2Ø 44 4F 53 2Ø 49 6E 74 65 72 6E 61 74 69 6F 6E
122: 61 6C ØØ ØA ØD 24 42 65 6C 69 65 62 69 67 65 2Ø
123: 54 61 73 74 65 2Ø 7A 75 6D 2Ø 53 74 61 72 74 2C
124: 2Ø ØØ 41 62 62 72 75 63 68 2Ø 6D 69 74 2Ø 45 53
125: 43 ØØ ØA ØD ØA ØD 24 41 6E 77 61 68 6C 3A 2Ø 5B
126: 24 18 19 1A 2C 2Ø 48 6F 6D 65 2C 2Ø 45 6E 64 5D
127: 2Ø 2Ø 2Ø ØØ 45 6E 64 65 3A 2Ø 5B 46 31 3Ø 5D 2Ø
128: 2Ø 41 62 62 72 75 63 68 3A 2Ø 5B 45 53 43 5D ØØ
129: ØA ØD ØA ØD 24 Ø6 53 26 8A 15 47 8Ø FA ØA 74 Ø2
13Ø: EB F5 52 32 FF B4 Ø3 57 CD 1Ø 5F 88 36 16 ØØ 5A
131: 89 3E 17 ØØ 89 3E 1A ØØ E8 EA ØØ 72 4D 89 1E 13
132: Ø9 26 8A 15 47 8Ø FA ØA 75 3B 8B 1E 13 Ø9 A1 1A
133: ØØ 8B CF 2B C8 FE C9 FE C9 FE C9 8Ø F9 41 76 Ø2
134: B1 41 88 4F 43 89 3E 1A ØØ E8 B9 ØØ 72 1C 8Ø 3E
135: 19 ØØ 15 75 ØA BA CØ Ø8 B4 Ø9 CD 21 F9 EB 4E 89
136: 1E 13 Ø9 EB BC 88 17 43 EB B7 8A 36 16 ØØ BF BB
137: ØØ 8A ØE 19 ØØ FE C9 32 ED 83 C7 44 FE C6 57 52
138: 51 F8 E8 1D Ø1 BA 1D ØØ B4 Ø9 CD 21 59 5A 5F E2
139: E8 BF BB ØØ 8A 36 16 ØØ F9 E8 Ø6 Ø1 53 32 FF B2
14Ø: ØC 8A 36 16 ØØ B4 Ø2 57 CD 1Ø 5F 5B F8 5B Ø7 C3
141: 2A 2A 2A 2Ø 46 65 68 6C 65 72 2Ø 43 46 47 45 44
142: 49 54 ØD ØA 2Ø 2Ø ØØ 2Ø 49 63 68 2Ø 6B 61 6E 6E
143: 2Ø 64 69 65 2Ø 45 6E 64 65 6D 61 72 6B 69 65 72
144: 75 6E 67 2Ø ØØ 45 4E 44 45 44 49 54 2E 53 59 53
145: 2Ø 6E 69 63 68 74 2Ø 66 69 6E 64 65 6E 21 ØØ Ø7
146: ØA ØD 24 ØØ ØØ 52 57 Ø6 B9 Ø5 ØØ BE 45 Ø9 8B 3E
147: 1A ØØ FC F3 A7 75 Ø6 Ø7 5F 5A F9 EB 17 BB BB ØØ
148: 32 E4 AØ 19 ØØ B9 44 ØØ F7 E1 Ø3 D8 FE Ø6 19 ØØ
149: Ø7 5F 5A F8 C3 44 45 4E 44 45 44 49 54 2E 53 59
15Ø: 53 8E Ø6 BF Ø6 8B 3E 17 ØØ BE BB ØØ BØ Ø1 32 ED
151: 8A 4C 43 FE C1 56 FC F3 A4 3A Ø6 19 ØØ 74 ØA 5E
152: 83 C6 44 FE CØ 47 47 EB E7 5E 53 32 FF B2 ØØ B6
153: 18 B4 Ø2 57 CD 1Ø 5F 5B BA 1D ØØ B4 Ø9 CD 21 BA
154: 95 Ø9 CD 21 C3 4B 6F 6E 66 69 67 75 72 61 74 69
155: 6F 6E 2Ø 67 65 84 6E 64 65 72 74 2E 2E 2E ØØ ØA
156: ØD 24 53 B3 7F 72 Ø2 B3 4E 32 FF 32 D2 52 B4 Ø2
157: 57 CD 1Ø 5F 8A Ø5 3C 44 75 Ø6 BA 2Ø ØØ EB 55 9Ø
158: 3C 46 75 Ø5 BA 2D ØØ EB 4B 3C 42 75 Ø5 BA 3A ØØ
159: EB 42 3C 4C 75 Ø5 BA 47 ØØ EB 39 3C 43 75 Ø5 BA
16Ø: 54 ØØ EB 3Ø 3C 51 75 Ø5 BA 61 ØØ EB 27 3C 5Ø 75
161: Ø5 BA 6E ØØ EB 1E 3C 53 75 Ø5 BA 88 ØØ EB 15 3C
162: 4B 75 Ø5 BA 95 ØØ EB ØC 3C 58 75 Ø5 BA 7B ØØ EB
163: Ø3 BA A2 ØØ B4 Ø9 CD 21 8A 4D 43 32 ED 47 5A B2
164: ØC 51 8A Ø5 B9 Ø1 ØØ B4 Ø9 57 CD 1Ø 5F FE C2 B4
165: Ø2 57 CD 1Ø 5F 47 59 E2 E8 5B C3 BB BB ØØ C6 Ø6
166: 1C ØØ Ø1 E8 7C Ø1 73 Ø4 F9 EB 3C 9Ø 8Ø FD Ø1 74
167: 25 5Ø 53 32 FF B3 7F B9 Ø1 ØØ B4 Ø9 57 CD 1Ø 5F
168: B4 Ø3 57 CD 1Ø 5F 8B FA 81 E7 FF ØØ 5B 58 88 41
169: F5 E8 Ø3 Ø1 EB CD 8Ø F9 ØA 74 ØC DØ E1 32 ED 8B
17Ø: F9 FF 95 AF ØØ EB BC C3 8A 36 16 ØØ 8B FB F8 E8
171: 1Ø FF AØ 19 ØØ 38 Ø6 1C ØØ 74 27 83 C3 44 8B FB
172: FE Ø6 16 ØØ 8A 36 16 ØØ F9 E8 F6 FE 53 32 FF B2
173: ØC 8A 36 16 ØØ B4 Ø2 57 CD 1Ø 5F 5B FE Ø6 1C ØØ
174: EB 2B C6 Ø6 1C ØØ Ø1 AØ 19 ØØ FE C8 28 Ø6 16 ØØ
175: 8A 36 16 ØØ BB BB ØØ 8B FB F9 E8 C5 FE 53 32 FF
176: B2 ØC 8A 36 16 ØØ B4 Ø2 57 CD 1Ø 5F 5B C3 8A 36
177: 16 ØØ 8B FB F8 E8 AA FE 8Ø 3E 1C ØØ Ø1 74 27 83
178: EB 44 8B FB FE ØE 16 ØØ 8A 36 16 ØØ F9 E8 92 FE
179: 53 32 FF B2 ØC 8A 36 16 ØØ B4 Ø2 57 CD 1Ø 5F 5B
18Ø: FE ØE 1C ØØ EB 2C AØ 19 ØØ A2 1C ØØ FE C8 ØØ Ø6
181: 16 ØØ 8A 36 16 ØØ B1 44 F6 E1 Ø3 D8 8B FB F9 E8
182: 6Ø FE 53 32 FF B2 ØC 8A 36 16 ØØ B4 Ø2 57 CD 1Ø
183: 5F 5B C3 53 32 FF B4 Ø3 57 CD 1Ø 5F 8Ø FA ØC 75
184: ØC 5B 8A 57 43 8Ø C2 ØB 53 32 FF EB Ø2 FE CA B4
185: Ø2 57 CD 1Ø 5F 5B C3 53 32 FF B4 Ø3 57 CD 1Ø 5F
186: 5B 53 8A 5F 43 8Ø C3 ØB 3A D3 75 Ø4 B2 ØC EB Ø2
187: FE C2 32 FF B4 Ø2 57 CD 1Ø 5F 5B C3 8A 57 43 8Ø
188: C2 ØB 8A 36 16 ØØ 53 32 FF B4 Ø2 57 CD 1Ø 5F 5B
189: C3 B2 ØC 8A 36 16 ØØ 53 32 FF B4 Ø2 57 CD 1Ø 5F
19Ø: 5B C3 53 32 E4 CD 16 3C ØØ 74 1D 3C 1B 75 Ø3 F9
191: EB 67 3C ØD 75 Ø2 EB 59 3C ØA 75 Ø2 EB 53 3C Ø8
192: 75 Ø2 EB 4D 32 ED EB 51 8Ø FC 44 75 Ø5 B1 ØA EB
193: 3A 9Ø 8Ø FC 48 75 Ø5 B1 Ø2 EB 3Ø 9Ø 8Ø FC 5Ø 75
194: Ø5 B1 Ø3 EB 26 9Ø 8Ø FC 4B 75 Ø5 32 C9 EB 1C 9Ø
195: 8Ø FC 4D 75 Ø5 B1 Ø1 EB 12 9Ø 8Ø FC 4F 75 Ø5 B1
196: Ø4 EB Ø8 9Ø 8Ø FC 47 75 Ø8 B1 Ø5 B5 Ø1 86 C4 EB
197: Ø8 B2 Ø7 B4 Ø2 CD 21 EB 8A 5B C3
```

Listing 1. »cfgedit.sys« binden Sie in die »config.sys« ein

```
[P14]     1: ; Programm: cfgedit.asm
[U16]     2: ; Funktion: Editieren der config.sys
[D14]     3: ; Srache: MASM 5.Ø, TASM 2.Ø
[AG3]     4: ; Autor: Roland Geier
[FU4]     5: ; (c)1991 DMV Widuch GmbH & Co. KG
[P3Ø]     6: ;
[W31]     7: PAGE 84
[D46]     8: TITLE  CFGEDIT 1.ØØ  (c) 199Ø Roland Geier
[MA1]     9: %NOMACS
[P3Ø]    1Ø: ;
[4N3]    11: ; Assemblieren/Linken:
```

Listing 2. Mit »cfgedit.asm« verändern Sie die »config.sys« während des Bootens

```
[463]   12: ; MASM / TASM CFGEDIT
[RD3]   13: ; LINK / TLINK CFGEDIT
[4K4]   14: ; EXE2BIN CFGEDIT CFGEDIT.SYS
[ØØØ]   15:
[PH5]   16: ; Konstanten, Makros, Strukturen
[ØØØ]   17:
[Z83]   18: wpt       equ  word ptr
[X53]   19: byp       equ  byte ptr
[LY2]   2Ø: ofs       equ  offset
[YS2]   21: sh        equ  short
[Z65]   22: Delay     equ  65ØØ  ; Verzögerung
[653]   23: _ESC      equ  27    ; ESC
[WV4]   24: CR        equ  13    ; Cariage Return
[LT3]   25: LF        equ  1Ø    ; Linefeed
[634]   26: BS        equ  8     ; Backspace
[7T4]   27: CrsUp     equ  Ø48h  ; erweiterte
[WD5]   28: CrsDn     equ  Ø5Øh  ; Tastaturcodes
[K56]   29: CrsLeft   equ  Ø4Bh  ; für Steuertasten
[HW2]   3Ø: CrsRight  equ  Ø4Dh
[BN2]   31: EndPos    equ  Ø4Fh
[SD2]   32: Home      equ  Ø47h
[V52]   33: F1Ø       equ  Ø44h
[TR4]   34: ; Bildschirmzeile und -spalte
[953]   35: ; für Editierstart
[LK2]   36: Y_Start   equ   4
[LM2]   37: X_Start   equ  12
[PH6]   38: INV_ATTR equ  Ø1111111b ; Invers-Attribut
[LF6]   39: NORMATTR equ  Ø1ØØ111Øb ; Normal-Attribut
[2X5]   4Ø: MAXENTRIES equ 2Ø ; max. editierbare
[MS6]   41: MAXLEN     equ 67 ; Zeilenanzahl und -länge
[ØØØ]   42:
[ØR6]   43: DOS  macro funcnr   ; Makro für DOS-Call
[172]   44:         mov ah,funcnr
[381]   45:         int 21h
[HPØ]   46:       endm
[ØØØ]   47:
[EG6]   48: VIDEO     macro funcnr   ; Makro für BIOS-
[R84]   49:            mov ah,funcnr ; Video-Call
[3B1]   5Ø:            push di
[Y71]   51:            int 1Øh
[G21]   52:            pop di
[HPØ]   53:           endm
[ØØØ]   54:
[5U6]   55: GotoXY    macro x,y       ; Makro zur Cursor-
[TL4]   56:             push bx       ; positionierung
[LN1]   57:             xor  bh,bh
[RG1]   58:             mov  dl,x
[9G1]   59:             mov  dh,y
[WA1]   6Ø:             VIDEO Ø2h
[T31]   61:             pop bx
[HPØ]   62:           endm
[ØØØ]   63:
[ZY5]   64: ; Struktur zur Aufnahme der Config-
[F14]   65: ; Zeilen und deren Länge
[ØØØ]   66:
[Y32]   67: Line      struc
[Y34]   68:             Content db MAXLEN dup (' ')
[XF1]   69:             Len     db Ø
[1V1]   7Ø: Line      ends
[ØØØ]   71:
[CM4]   72: Code      Segment Public 'Code'
[H65]   73:           Assume cs:code, ds:code, es:code
[GTØ]   74:           org Ø
[P61]   75: ; Daten
[MW6]   76: Start:    dd ØFFFFFFFFh  ; Verbindg. Treiber
[1Z4]   77:           dw 8ØØØh       ; Zeichenorientiert
[DZ4]   78:           dw strat       ; Zeiger auf Strat
[ST4]   79:           dw Intr        ; Zeiger auf Intr
[WF4]   8Ø:           db 'CONXXXX$'  ; Treibername
[WB4]   81: Save_bx   dw ?           ; für BX
[SB4]   82: Save_es   dw ?           ; für ES
[RC5]   83: y_pos     db Ø ; y-pos Editierzeile
[UE7]   84: StrStart  dw Ø ; Adresse 1.String 1.Struktur
[W57]   85: LineCnt   db Ø ; insg. zu editierende Zeilen
[EK6]   86: Marker    dw Ø ; Marke für proc ListConfig
[C96]   87: Selected  db Ø ; Aktuelle Editier-Zeile
[HH2]   88: CRLF      db 1Ø,13,'$'
[NP3]   89: D_Str     db 'DEVICE    = $'
[TL3]   9Ø: F_Str     db 'FILES     = $'
[2V3]   91: B_Str     db 'BUFFERS   = $'
[V14]   92: L_Str     db 'LASTDRIVE = $'
[4L3]   93: C_Str     db 'BREAK     = $'
[XX3]   94: Q_Str     db 'COUNTRY   = $'
[HY3]   95: P_Str     db 'DRIVPARM  = $'
[YF3]   96: X_Str     db 'FCBS      = $'
[8N3]   97: S_Str     db 'SHELL     = $'
[VV3]   98: K_Str     db 'STACKS    = $'
[8E4]   99: _Unknown db 'UNBEKANNT = $'
[ØØØ]  1ØØ:
[1X6]  1Ø1: Jmp_Tab   dw ofs KeyLeft  ; Sprungtabelle
[US4]  1Ø2:           dw ofs KeyRight ; für Cursor-
[984]  1Ø3:           dw ofs KeyUp    ; bewegungen
[ZW1]  1Ø4:           dw ofs KeyDn
[B12]  1Ø5:           dw ofs KeyEnd
[T82]  1Ø6:           dw ofs KeyHome
[ØØØ]  1Ø7:
[XU6]  1Ø8: ; Strukturen für die Aufnahme der in das
[F16]  1Ø9: ; RAM gelesenen Zeilen der config.sys
[ZY6]  11Ø: ; und deren Längen; die letzte Struktur
[7T6]  111: ; dient als Überlaufpuffer bei einer zu
[C92]  112: ; langen Zeile
```

```
[ØØØ]  113:
[DF5]  114: Lines     Line  MAXENTRIES+1 dup (<>)
[ØØØ]  115:
[9W6]  116: ; Strat  : Strategy-Routine des Treibers
[XL2]  117: Strat     proc far
[G93]  118:           mov  cs:save_bx,bx
[W83]  119:           mov  cs:save_es,es
[3PØ]  12Ø:           ret
[YZ1]  121: Strat     endp
[ØØØ]  122:
[HK6]  123: ; Intr: Interrupt-Routine des Treibers
[TS5]  124: ;  Ruft die einzelnen Routinen auf,
[BF6]  125: ;  informiert DOS bei Ende der Ausführung
[8Ø5]  126: ;  und teilt die nächste freie
[WG3]  127: ;  Speicheradresse mit
[ØØØ]  128:
[8G2]  129: Intr      proc far
[SV4]  13Ø:            push ax    ; wichtige Register
[K73]  131:            push bx    ; sichern
[WC1]  132:            push cx
[ZC1]  133:            push dx
[BC1]  134:            push ds
[SH4]  135:            pushf      ; Sichern der Flags
[7Z2]  136:            push cs    ; ds = cs
[5A1]  137:            pop  ds
[5E5]  138:            mov  es,Save_es ; Adr Datenblock
[KA5]  139:            mov  bx,Save_bx ; restaurieren
[5H4]  14Ø:            ; Initialisierungsroutine
[9G1]  141:            call init
[5K3]  142:            ; Carry-Flag gesetzt?
[FA2]  143:            jc   sh Do_Nix
[G35]  144:            ; es:di->Adresse, in der Inhalt
[KK5]  145:            ; der config.sys eingelesen wurde
[HØ3]  146:            mov  di,es:[bx+18]
[N13]  147:            mov  ax,es:[bx+2Ø]
[GS1]  148:            mov  es,ax
[WH5]  149:            mov  ConfRam_Seg,ax ; es peichern
[YV5]  15Ø:            call ListConfig  ; Einträge listen
[YY3]  151:            jc   sh Do_Nix   ; Fehler?
[KK5]  152:            call Handler ; Einträge editieren
[T92]  153:            jnc   sh _Move
[3U1]  154:            GotoXY Ø,24
[1E4]  155:            ; Abbruchsmeldung ausgeben
[A63]  156:            mov  dx,ofs ESCMsg
[X61]  157:            DOS  Ø9h
[H15]  158:            jmp  sh Do_Nix ; Nichts Moven
[AS5]  159:            ; Struktureinträge CONTENT wieder
[E63]  16Ø:            ; ins RAM schreiben
[YU2]  161: _Move:     call Move
[Ø17]  162: Do_Nix:    ; Erste freie Speicherstelle nach
[W85]  163:            ; cfgedit.sys in die Offsets 14-17
[8L5]  164:            ; des Device-Datenblocks eintragen
[XB5]  165:            ; Da CFGEDIT nach dem Booten nicht
[AE5]  166:            ; mehr benötigt wird, kann sein
[7F5]  167:            ; Speicher durch Angabe der Label-
[DB5]  168:            ; adresse von START freigegeben
[BC1]  169:            ; werden!
[5V2]  17Ø:            mov  es,Save_es
[VV2]  171:            mov  bx,Save_bx
[E45]  172:            mov  wpt es:[bx+14],ofs start
[RW3]  173:            mov  wpt es:[bx+16],cs
[CP5]  174:            ;Ready-Code über den Device-Daten-
[U24]  175:            ;block an DOS übergeben
[ØS3]  176:            or   wpt es:[bx+3],Ø1ØØh
[ØL4]  177:            popf       ; Flags und Register
[7R4]  178:            pop  ds    ; wiederherstellen
[TA1]  179:            pop  dx
[PA1]  18Ø:            pop  cx
[LA1]  181:            pop  bx
[GA1]  182:            pop  ax
[3PØ]  183:            ret
[TP2]  184: ConfRam_Seg dw Ø
[5M5]  185: ESCMsg     db 'CFGEDIT abgebrochen...',
[Ø22]  186:            db 1Ø,13,1Ø,13,'$'
[9W1]  187: Intr       endp
[ØØØ]  188:
[HU6]  189: ; Init: Ausgabe des Copyright und Warten
[S56]  19Ø: ;       auf Tastendruck bzw. auf Ablauf
[ZØ3]  191: ;       der Wartezeit
[H36]  192: ; Ausgabe: Carry-Flag = 1: Abbruch durch
[EZ5]  193: ;           Esc oder Ablauf der Wartezeit
[WU5]  194: ;          Carry-Flag = Ø: Taste Betätigt
[PP2]  195: Init       proc near
[EC1]  196:            push es
[TC1]  197:            push bx
[YC3]  198:            mov  y_pos,Y_Start
[5V2]  199:            mov  dx,ofs CRLF
[X61]  2ØØ:            DOS  Ø9h
[313]  2Ø1:            mov  dx,ofs CopyR
[E61]  2Ø2:            int  21h
[2T2]  2Ø3:            mov  dx,ofs Msg
[6R4]  2Ø4:            int  21h       ; Meldung ausgeben
[WV1]  2Ø5:            mov  cx,4ØØØ
[2D6]  2Ø6: @LoopL:    DOS  ØBh       ; Eingabestatus lesn
[965]  2Ø7:            cmp  al,ØFFh ; Zeichen vorhanden?
[L25]  2Ø8:            jnz  sh No_Key ; Nein -> Hüpf!
[CA4]  2Ø9:            DOS  Ø7h       ; Ja -> auslesen
[NT4]  21Ø:            cmp  al,_ESC   ; Zeichen = ESC?
[ZB4]  211:            jz   sh Ch_ESC ; Ja -> Hüpf!
[DL1]  212:            GotoXY Ø,Ø
[SV1]  213:            mov  cx,2ØØØ
```

```
[LN1]  214:          xor  bh,bh
[ZW2]  215:          mov  bl,NORMATTR
[UK1]  216:          mov  al,32
[Y51]  217:          VIDEO 9
[313]  218:          mov  dx,ofs CopyR
[X61]  219:          DOS  Ø9h
[MX2]  22Ø:          mov  dx,ofs Help1
[E61]  221:          int  21h
[PF1]  222:          mov  cx,1
[7L1]  223:          mov  al,27
[3B1]  224:          VIDEO Ø9h
[XA1]  225:          VIDEO Ø3h
[451]  226:          inc   dl
[WA1]  227:          VIDEO Ø2h
[PX2]  228:          mov  dx,ofs Help2
[N41]  229:          DOS  Ø9
[XW4]  23Ø:          clc          ; Carryflag löschen
[VC5]  231:          jmp  sh InitEnd ; und Ende, sonst
[686]  232: Ch_ESC:  stc            ; Carryflag setzen
[584]  233:          jmp  sh InitEnd ; und Ende
[9V2]  234: No_Key:  cmp  cx,1
[Z52]  235:          jnz  sh Again
[EP4]  236:          ; TimeOut -> Carryflag setzen
[ENØ]  237:          stc
[CP2]  238:          jmp  sh InitEnd
[236]  239: Again:   loop @LoopL     ; dec cx, nochmal!
[MP2]  24Ø: InitEnd: pop bx
[G31]  241:          pop es
[3PØ]  242:          ret
[3Z4]  243: CopyR    db 'CFGEDIT 1.ØØ Roland',
[3Ø5]  244:          db ' Geier & DOS International',
[YA1]  245:          db 1Ø,13,'$'
[5D6]  246: Msg      db 'Beliebige Taste zum Start, ',
[X93]  247:          db 'Abbruch mit ESC',
[Ø22]  248:          db 1Ø,13,1Ø,13,'$'
[1F3]  249: Help1    db 'Anwahl: [$'
[Y55]  25Ø: Help2    db  24,25,26,', Home, End]   ',
[N75]  251:          db 'Ende: [F1Ø]  Abbruch: [ESC]',
[Ø22]  252:          db 1Ø,13,1Ø,13,'$'
[ZU1]  253: Init     endp
[ØØØ]  254:
[C37]  255: ; ListConfig: Auslesen d.CONFIG.SYS-Zeilen
[BH5]  256: ;             aus dem RAM und Belegung der
[CD5]  257: ;             Strukturvariablen Lines
[M95]  258: ; Ausgabe   : Carry-Flag = 1: Fehler
[KB5]  259: ;             Carry-Flag = Ø: kein Fehler
[ØØØ]  26Ø:
[TD3]  261: ListConfig proc near
[EC1]  262:          push es
[TC1]  263:          push bx
[UB4]  264:          ; Ersten Eintrag im Buffer
[A34]  265:          ; (CFGEDIT.SYS) weglassen!
[7C7]  266: Next_Ch: mov  dl,byp es:[di] ;Zeichen holen
[YD5]  267:          inc  di    ; di->nächstes Zeichen
[HY3]  268:          cmp  dl,LF ; Zeichen = LF?
[MV2]  269:          jz   sh _1stEntry
[US2]  27Ø:          jmp  sh Next_Ch
[7V1]  271: _1stEntry:
[ZC1]  272:          push dx
[LN1]  273:          xor  bh,bh
[YØ5]  274:          ; Cursorposition feststellen
[XA1]  275:          VIDEO Ø3h
[235]  276:          mov  y_pos,dh    ; y_pos merken
[TA1]  277:          pop  dx
[ØK4]  278:          ; Position 1. Eintrag merken
[RU2]  279:          mov  StrStart,di
[ØX4]  28Ø:          ; Marker = 1. Zeichen, 1. Eintrag
[5M2]  281:          mov  Marker, di
[W75]  282:          ; Überprüfung auf ENDEDIT.SYS
[DD2]  283:          call CheckLast
[YP4]  284:          ; Letzter Eintrag gefunden?
[VW1]  285:          jc   EndList
[AV2]  286:          mov  StrucSav,bx
[CL6]  287: LoopL:   mov  dl,byp es:[di] ; Zeichen->dl
[YD5]  288:          inc  di    ; di->nächstes Zeichen
[HY3]  289:          cmp  dl,LF ; Zeichen = LF?
[732]  29Ø:          jnz  Weiter
[X12]  291:          ; LF entdeckt
[YD5]  292:          ; Strukturpointer restaurieren
[745]  293:          ; und Strukturvariable belegen
[9X2]  294:          mov  bx,StrucSav
[YF2]  295:          mov  ax,Marker
[RR1]  296:          mov  cx,di
[CR1]  297:          sub  cx,ax
[TR4]  298:          ; Länge nun in cl, incl. CR+LF
[P45]  299:          dec  cl ; CR und LF sowie das als
[CE4]  3ØØ:          dec  cl ; Kennung dienende 1.
[EM5]  3Ø1:          dec  cl ; Zeichen nicht mitzählen
[MA5]  3Ø2:          cmp  cl,MAXLEN-2 ; Zeile zu lang?
[Z42]  3Ø3:          jna  sh LenOk
[UV3]  3Ø4:          ; CR + 1.Zeichen beachten!
[7P2]  3Ø5:          mov  cl,MAXLEN-2
[X15]  3Ø6:          ; Strukturvariable LEN belegen
[6Y3]  3Ø7: LenOk:   mov  [bx.Len],cl
[Ø64]  3Ø8:          ; Marker = Eintrag nach CR!
[SD2]  3Ø9:          mov  Marker,di
[DD2]  31Ø: call CheckLast
[CA5]  311:          ; Carry-Flag -> letzter Eintrag
[GZ2]  312:          ; bereits gefunden
[1H2]  313:          jc   sh EndList
```

```
[F96]  314:          ; Überprüfung auf Überschreitung
[RC5]  315:          ; der Editiergrenze von 2Ø Zeilen
[NØ5]  316:          ; bzw. auf Fehlen der Endemarkie-
[DX2]  317:          ; rung ENDEDIT.SYS
[D94]  318:          cmp  LineCnt,MAXENTRIES+1
[YR1]  319:          jnz  sh OK
[E13]  32Ø:          mov  dx,ofs Error
[VE5]  321:          DOS  Ø9h  ; Fehlermeldung ausgeben
[L94]  322:          stc       ; Carry-Flag setzen
[AB4]  323:          jmp  sh Stop    ; ...und Ende
[M64]  324: OK:      ; Pointer bx sichern
[AV2]  325:          mov  StrucSav,bx
[C64]  326:          ; (bx->aktuelle Struktur)
[X72]  327:          jmp  sh LoopL
[N46]  328: Weiter:  ; Zeichen in Struktur ablegen
[XL3]  329:          mov [bx.Content],dl
[ZM4]  33Ø:          ; Strukturpointer erhöhen
[E81]  331:          inc  bx
[5B4]  332:          ; ...und nächstes Zeichen
[X72]  333:          jmp  sh LoopL
[WU3]  334: EndList: mov  dh,y_pos
[E55]  335:          ; di mit erster Struktur laden
[DY2]  336:          mov  di,ofs Lines
[473]  337:          ; Anzahl Zeilen -> cl
[MS2]  338:          mov  cl,LineCnt
[4S4]  339:          ; erste Zeile nicht ausgeben
[W51]  34Ø:          dec  cl
[SN1]  341:          xor  ch,ch
[DK6]  342: lp:      add  di,SIZE Line ; nächste Zeile
[5A4]  343:          ; nächste Bildschirmzeile
[M61]  344:          inc  dh
[3B1]  345:          push di
[ZC1]  346:          push dx
[WC1]  347:          push cx
[PLØ]  348:          clc
[2C5]  349:          ; Zeile normal ausgeben und ein
[RZ2]  35Ø:          ; CRLF hinterher...
[A42]  351:          call OutLine
[5V2]  352:          mov  dx,ofs CRLF
[N41]  353:          DOS  Ø9
[PA1]  354:          pop  cx
[TA1]  355:          pop  dx
[X81]  356:          pop  di
[CK5]  357:          loop lp       ; zur nächsten Zeile
[F23]  358:          ; di -> erste Zeile
[DY2]  359:          mov  di,ofs Lines
[VA2]  36Ø:          mov  dh,y_pos
[ENØ]  361:          stc
[DM4]  362:          ; 1. Eintrag invers ausgeben
[A42]  363:          call OutLine
[VP3]  364:          GotoXY X_Start,y_pos
[PW3]  365:          ; kein Fehler-kein Carry!
[PLØ]  366:          clc
[YA2]  367: Stop:    pop  bx
[8A1]  368:          pop  es
[3PØ]  369:          ret
[ZG5]  37Ø: Error   db '*** Fehler CFGEDIT',13,1Ø,'  ',
[TD5]  371:         db ' Ich kann die Endemarkierung ',
[115]  372:         db 'ENDEDIT.SYS nicht finden!',
[BL1]  373:         db 7,1Ø,13,'$'
[Ø82]  374: StrucSav dw Ø
[ØØØ]  375:
[SP2]  376: ListConfig endp
[ØØØ]  377:
[ØT6]  378: ; CheckLast: prüft auf Zeile dendedit.sys
[VP3]  379: ;             belegt LineCnt
[LK6]  38Ø: ; Ausgabe : bx Zeiger auf Zeilenstruktur
[ØØØ]  381:
[7C2]  382: CheckLast proc
[SB4]  383:          push dx   ; in dl steht CR!!!
[3B1]  384:          push di
[EC1]  385:          push es
[6P5]  386:          ; Länge EndStr in Words (Bedingung
[KR1]  387:          ; für Ende!)
[YF1]  388:          mov  cx,5
[VD4]  389:          ; Laden Adresse 1.+ 2. String
[9C2]  39Ø:          lea  si,EndStr
[BE2]  391:          mov  di,Marker
[DC4]  392:          ; Richtungsflag löschen
[ULØ]  393:          cld
[2K5]  394:     repe cmps [di],Marker ; Vergleich
[KZ5]  395:          jnz  sh NotLast ; Letzter Eintrag?
[8A1]  396:          pop  es
[X81]  397:          pop  di
[TA1]  398:          pop  dx
[SE5]  399:          ; Carry-Flag setzen bei letztem
[2F1]  4ØØ:          ; Eintrag
[ENØ]  4Ø1:          stc
[9T4]  4Ø2:          jmp  sh EndChck     ; Ja -> Ende!
[TE6]  4Ø3: NotLast: ; Struktur für Zeile laden
[3Ø3]  4Ø4:          mov  bx,ofs Lines
[EN1]  4Ø5:          xor  ah,ah
[UY4]  4Ø6:          ; al->CONFIG.SYS-Zeilennummer
[XF3]  4Ø7:          mov  al,byp LineCnt
[CP4]  4Ø8:          ; Strukturgröße nach cx
[2Z2]  4Ø9:          mov  cx,SIZE Line
[244]  41Ø:          mul  cx              ; ax = ax * cx
[NG5]  411:          ; bx=(SIZE Line)*LineCnt+OFS Lines
```

Listing 2. Mit »cfgedit.asm« verändern Sie die »config.sys« während des Bootens

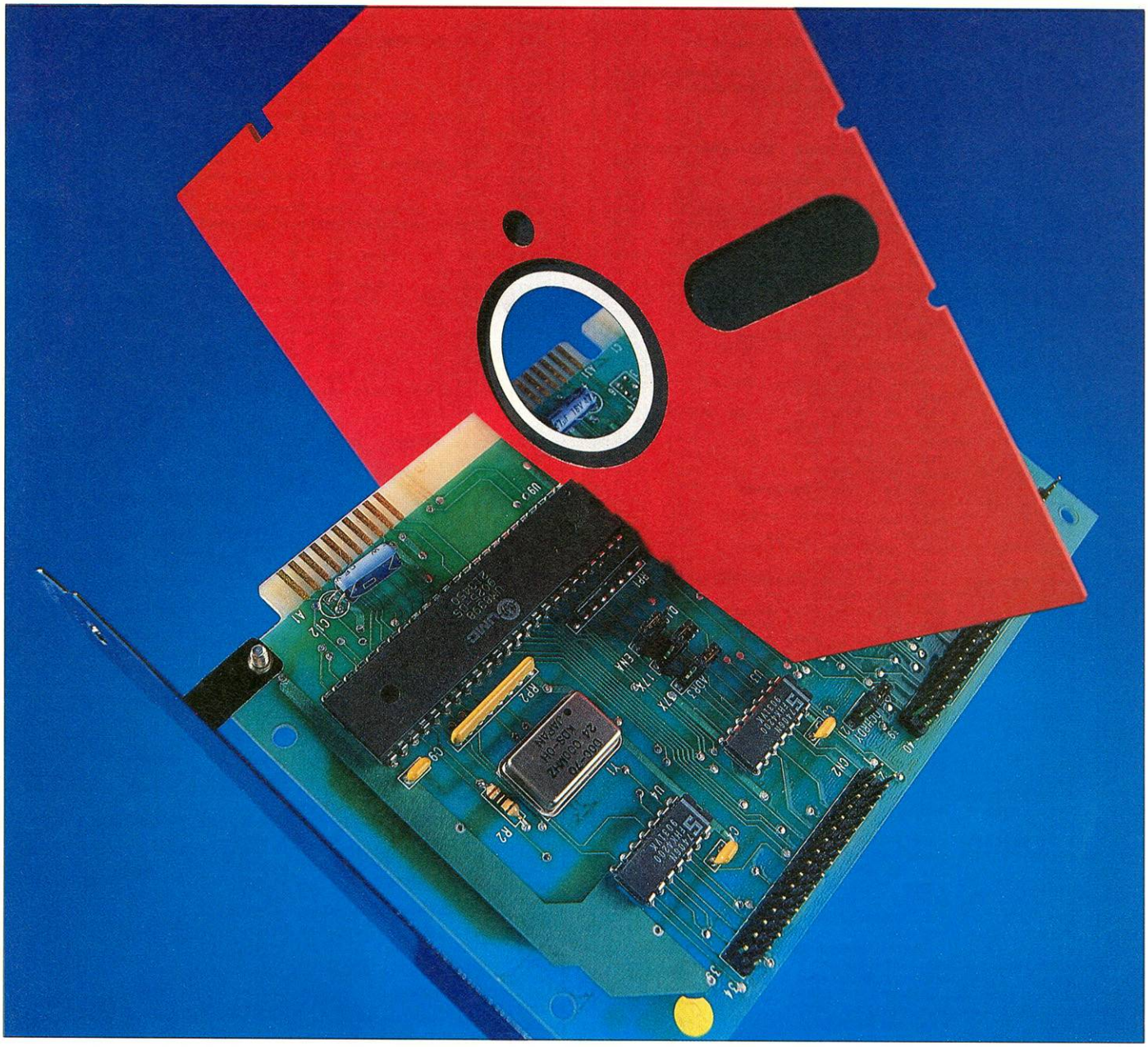
```
[UM1]  412:           add  bx,ax
[PD4]  413:           ; Eintragszähler erhöhen
[XØ2]  414:           inc  LineCnt
[8A1]  415:           pop  es
[X81]  416:           pop  di
[TA1]  417:           pop  dx
[725]  418:           ; Carry-Flag löschen da nicht
[6X2]  419:           ; letzter Eintrag
[PLØ]  42Ø:           clc
[412]  421: EndChck: ret
[734]  422: EndStr   db 'DENDEDIT.SYS'
[VF2]  423: CheckLast  endp
[ØØØ]  424:
[5C7]  425: ; Move: Kopiert Struktureinträge Content
[XC4]  426: ;       zurück in den Speicher
[ØØØ]  427:
[HX1]  428: Move      proc
[2G3]  429:           mov  es,ConfRam_Seg
[M95]  43Ø:           ; es:di->config.sys im Speicher
[ZW2]  431:           mov  di,StrStart
[A52]  432:           lea  si,Lines
[ØE1]  433:           mov  al,1
[SN1]  434:           xor  ch,ch
[N84]  435: NxtLine: mov  cl,[si.Len]
[DØ5]  436:           ; Originallänge restaurieren
[171]  437:           inc  cl
[GC1]  438:           push si
[ULØ]  439:           cld
[ZY4]  44Ø:     rep   movsb ; Zeile verschieben
[GC5]  441:           cmp  al,LineCnt ; schon fertig?
[4N4]  442:           jz   sh EndMove ; Ja -> Hüpf!
[AA1]  443:           pop  si
[RH3]  444:           ; si -> nächste Zeile
[NU2]  445:           add  si,SIZE Line
[ØH4]  446:           ; Zeilenzähler al erhöhen
[V61]  447:           inc  al
[YM3]  448:           ; Platz für CR und LF!
[S61]  449:           inc  di
[S61]  45Ø:           inc  di
[4T2]  451:           jmp  sh NxtLine
[SP2]  452: EndMove: pop si
[EZ4]  453:           ; Änderungsmeldung ausgeben
[3U1]  454:           Gotoxy Ø,24
[5V2]  455:           mov  dx,ofs CRLF
[X61]  456:           DOS  Ø9h
[Z53]  457:           mov  dx,ofs EndMsg
[E61]  458:           int  21h
[3PØ]  459:           ret
[WK6]  46Ø: EndMsg   db 'Konfiguration geändert...',
[YA1]  461:           db 1Ø,13,'$'
[FW1]  462: Move      endp
[ØØØ]  463:
[RG6]  464: LOCALS    ; Vermeidet Label-Katastrophen!
[ØØØ]  465:
[7R7]  466: ; OutLine: Gibt Struktureintrag Content aus
[3G6]  467: ; Eingabe: Carry-Flag = 1: Inverse Ausgabe
[HT5]  468: ;          Carry-Flag = Ø: Normale Ausgabe
[EB2]  469: ;          dh: y_pos
[HX5]  47Ø: ;          es:di: Zeiger auf Zeichenkette
[ØØØ]  471:
[P92]  472: OutLine   proc
[TC1]  473:           push bx
[1X5]  474:           mov  bl,INV_ATTR ; Invers-Attribut
[MY1]  475:           jc   sh @@Inv
[ZW2]  476:           mov  bl,NORMATTR
[726]  477: @@Inv:    xor  bh,bh  ; Bildschirmseite Ø
[TP1]  478:           xor  dl,dl
[ZC1]  479:           push dx
[D74]  48Ø:           ; Cursorposition setzen
[C61]  481:           VIDEO Ø2
[WX4]  482:           ; ersten Buchstaben nach al ->
[BL3]  483:           ; zeigt Befehlsart an!
[GU2]  484:           mov  al,byp [di]
[2R1]  485:           cmp  al,'D'
[R75]  486:           jnz  sh @@ChckF   ; Durchchecken
[7A4]  487:           mov  dx,ofs D_Str ; aller
[PS4]  488:           jmp  @@Inv1       ; Anfangsbuch-
[6Y4]  489: @@ChckF: cmp  al,'F'       ; staben
[2M2]  49Ø:           jnz  sh @@ChckB
[Ø33]  491:           mov  dx,ofs F_Str
[5A2]  492:           jmp  sh @@Inv1
[PX2]  493: @@ChckB: cmp  al,'B'
[AN2]  494:           jnz  sh @@ChckL
[S23]  495:           mov  dx,ofs B_Str
[5A2]  496:           jmp  sh @@Inv1
[6Ø3]  497: @@ChckL: cmp  al,'L'
[6M2]  498:           jnz  sh @@ChckC
[C33]  499:           mov  dx,ofs L_Str
[5A2]  5ØØ:           jmp  sh @@Inv1
[YX2]  5Ø1: @@ChckC: cmp  al,'C'
[YN2]  5Ø2:           jnz  sh @@ChckQ
[U23]  5Ø3:           mov  dx,ofs C_Str
[5A2]  5Ø4:           jmp  sh @@Inv1
[E13]  5Ø5: @@ChckQ: cmp  al,'Q'
[UN2]  5Ø6:           jnz  sh @@ChckP
[P33]  5Ø7:           mov  dx,ofs Q_Str
[5A2]  5Ø8:           jmp  sh @@Inv1
[613]  5Ø9: @@ChckP: cmp  al,'P'
[6P2]  51Ø:           jnz  sh @@ChckS
[M33]  511:           mov  dx,ofs P_Str
[5A2]  512:           jmp  sh @@Inv1
```

```
[Y13]  513: @@ChckS: cmp  al,'S'
[6N2]  514:           jnz  sh @@ChckK
[U33]  515:           mov  dx,ofs S_Str
[5A2]  516:           jmp  sh @@Inv1
[YZ2]  517: @@ChckK: cmp  al,'K'
[UP2]  518:           jnz  sh @@ChckX
[A33]  519:           mov  dx,ofs K_Str
[5A2]  52Ø:           jmp  sh @@Inv1
[633]  521: @@ChckX: cmp  al,'X'
[GØ3]  522:           jnz  sh @@Unknown
[443]  523:           mov  dx,ofs X_Str
[5A2]  524:           jmp  sh @@Inv1
[T45]  525: @@Unknown: mov dx,ofs _Unknown
[N75]  526: @@Inv1:   ; Befehlsnamen ausgeben
[X61]  527:           DOS  Ø9h
[MT2]  528:           mov  cl,[di.Len]
[SN1]  529:           xor  ch,ch
[8W3]  53Ø:           ; di -> zweites Zeichen
[S61]  531:           inc  di
[TA1]  532:           pop  dx
[WW2]  533:           mov  dl,X_Start
[7H2]  534: @@Inv2:   push cx
[7G5]  535:           mov  al,byp [di] ; Zeichen nach al
[PF1]  536:           mov  cx,1
[PH4]  537:           VIDEO Ø9h ; Zeichen ausgeben
[UB4]  538:           inc  dl   ; Cursor 1 rechts
[5F5]  539:           VIDEO Ø2h ; Funk Ø2: Cursor Setzen
[2E5]  54Ø:           inc  di   ; di ->nächstes Zeichen
[PA1]  541:           pop  cx
[XU4]  542:           loop @@Inv2   ; und von vorne!
[LA1]  543:           pop  bx
[3PØ]  544:           ret
[M82]  545: OutLine   endp
[ØØØ]  546:
[VZ6]  547: ; Handler: Auswerten von Tastatureingaben
[ØØØ]  548:
[B72]  549: Handler   proc
[2M3]  55Ø:           ; bx -> erste Struktur
[3Ø3]  551:           mov  bx,ofs Lines
[9Z3]  552:           ; erste Zeile editieren
[GK2]  553:           mov  Selected,1
[283]  554: @@Again: call GetKey
[TL2]  555:           jnc  sh @@NoESC
[ENØ]  556:           stc
[Y84]  557:           jmp  @@End     ; ESC = Abbruch
[LM5]  558: @@NoESC: cmp  ch,1       ; Steuertaste?
[M34]  559:           jz   sh @@Crs   ; Ja -> Hüpf
[9L2]  56Ø: ; normale Taste
[PC1]  561:           push ax
[TC1]  562:           push bx
[LN1]  563:           xor  bh,bh
[MX2]  564:           mov  bl,INV_ATTR
[PF1]  565:           mov  cx,1
[BB5]  566:           ; Zeichen invers auf Bildschirm
[GG5]  567:           ; ausgeben und Cursorpos auslesen
[3B1]  568:           VIDEO Ø9h
[XA1]  569:           VIDEO Ø3h
[BR1]  57Ø:           mov  di,dx
[PY1]  571:           and  di,ØØFFh
[LA1]  572:           pop  bx
[GA1]  573:           pop  ax
[216]  574:           ; ASCII-Code in Struktur verändern
[VN5]  575:           mov  [bx.Content+di+1-X_Start],al
[175]  576:           call KeyRight  ; um eins rechts
[C55]  577:           jmp  @@Again   ; neue Taste holen
[U72]  578: ; Steuertaste
[CB5]  579: @@Crs:    cmp  cl,1Ø     ; F1Ø für Ende?
[5Z1]  58Ø:           jz   sh @@End
[YU6]  581: ; andere Tasten über Tabelle anspringen
[NC1]  582:           shl  cl,1
[SN1]  583:           xor  ch,ch
[9R1]  584:           mov  di,cx
[ØB3]  585:           call [Jmp_Tab + di]
[ZY1]  586:           jmp  @@Again
[PS1]  587: @@End:    ret
[962]  588: Handler   endp
[ØØØ]  589:
[FN5]  59Ø: ; KeyUp,-Dn,-Left,-Right,-End,-Home:
[S56]  591: ;         Steuerung der Cursorbewegungen
[ØØØ]  592:
[VU4]  593: KeyDn     proc           ; Down-Taste
[VA2]  594:           mov  dh,y_pos
[7R1]  595:           mov  di,bx
[PLØ]  596:           clc
[1Ø4]  597:           ; Zeile normal ausgeben
[A42]  598:           call OutLine
[ES2]  599:           mov  al,LineCnt
[FA5]  6ØØ:           cmp  Selected,al ; letzten Zeile?
[6D4]  6Ø1:           jz   sh @@Wrap   ; Ja -> Hüpf
[923]  6Ø2:           add  bx,(SIZE Line)
[V54]  6Ø3:           ; Zeiger auf Zeile nach di
[7R1]  6Ø4:           mov  di,bx
[S44]  6Ø5:           ; eine Zeile weiter unten
[PU1]  6Ø6:           inc  y_pos
[VA2]  6Ø7:           mov  dh,y_pos
[ENØ]  6Ø8:           stc
[L14]  6Ø9:           ; Zeile invers ausgeben
[A42]  61Ø:           call OutLine
[VP3]  611:           GotoXY X_Start,y_pos
[P24]  612:           ; Selected aktualisieren
[H52]  613:           inc  Selected
```

```
[PL2]  614:           jmp  sh @@DnEnd
[F76]  615: @@Wrap:   ; Cursor in der letzten Zeile
[GK2]  616:           mov  Selected,1
[PP5]  617:           mov  al,LineCnt  ; y_pos berechnen
[ØZ4]  618:           ; al (=LineCnt) um 1 vermindern
[P51]  619:           dec  al
[482]  62Ø:           sub  y_pos,al
[VA2]  621:           mov  dh,y_pos
[WN5]  622:           mov  bx,ofs Lines ;bx->1.Struktur
[7R1]  623:           mov  di,bx
[L14]  624:           ; Zeile invers ausgeben
[ENØ]  625:           stc
[A42]  626:           call OutLine
[VP3]  627:           GotoXY X_Start,y_pos
[UZ1]  628: @@DnEnd:  ret
[MY1]  629: KeyDn     endp
[ØØØ]  63Ø:
[9X3]  631: KeyUp     proc   ; Up-Taste
[VA2]  632:           mov  dh,y_pos
[7R1]  633:           mov  di,bx
[1Ø4]  634:           ; Zeile normal ausgeben
[PLØ]  635:           clc
[A42]  636:           call OutLine
[G24]  637:           ; sind wir in der 1. Zeile?
[YG2]  638:           cmp  Selected,1
[XR4]  639:           jz   sh @@Wrap      ; Ja -> Hüpf
[S43]  64Ø:           sub  bx,(SIZE Line)
[7R1]  641:           mov  di,bx
[HT1]  642:           dec  y_pos
[VA2]  643:           mov  dh,y_pos
[L14]  644:           ; Zeile invers ausgeben
[ENØ]  645:           stc
[A42]  646:           call OutLine
[VP3]  647:           GotoXY X_Start,y_pos
[C42]  648:           dec  Selected
[WN2]  649:           jmp  sh @@UpEnd
[N16]  65Ø: @@Wrap:   ; Cursor in der ersten Zeile
[ES2]  651:           mov  al,LineCnt
[9B5]  652:           mov  Selected,al ; Berechnung der
[XD4]  653:           dec  al          ; Strukturadr.
[CL4]  654:           add  y_pos,al    ; der letzten
[2F5]  655:           mov  dh,y_pos    ; zu editierenden
[AE4]  656:           mov  cl,(SIZE Line) ; Zeile
[291]  657:           mul  cl
[UM1]  658:           add  bx,ax
[7R1]  659:           mov  di,bx
[L14]  66Ø:           ; Zeile invers ausgeben
[ENØ]  661:           stc
[A42]  662:           call OutLine
[VP3]  663:           GotoXY X_Start,y_pos
[Ø22]  664: @@UpEnd:  ret
[9Z1]  665: KeyUp     endp
[ØØØ]  666:
[425]  667: KeyLeft   proc     ; Links-Taste
[TC1]  668:           push bx
[LN1]  669:           xor  bh,bh
[C65]  67Ø:           VIDEO Ø3h   ;CrsPos feststellen
[Z36]  671:           cmp  dl,X_Start ;x_pos ganz links?
[UW4]  672:           jnz  sh @@LftOk ; Nein -> Hüpf
[LA1]  673:           pop  bx
[UP5]  674:           ; letzte x_pos der Zeile berechnen
[DV2]  675:           mov  dl,[bx.Len]
[AX2]  676:           add  dl,X_Start-1
[TC1]  677:           push bx
[LN1]  678:           xor  bh,bh
[KN2]  679:           jmp  sh @@EndLt
[GK2]  68Ø: @@LftOk:  dec dl
[7U5]  681: @@EndLt:  VIDEO Ø2h    ; Cursor setzen
[T31]  682:           pop bx
[3PØ]  683:           ret
[C82]  684: KeyLeft   endp
[ØØØ]  685:
[FD5]  686: KeyRight  proc          ; Rechts-Taste
[TC1]  687:           push bx
[LN1]  688:           xor  bh,bh
[H55]  689:           VIDEO Ø3h  ; CrsPos feststellen
[LA1]  69Ø:           pop  bx
[TC1]  691:           push bx
[7V2]  692:           mov  bl,[bx.Len]
[4X2]  693:           add  bl,X_Start-1
[L95]  694:           cmp  dl,bl   ; Cursor ganz rechts?
[Y15]  695:           jnz  sh @@RtOk      ; Nein -> hüpf
[WW2]  696:           mov  dl,X_Start
[4P2]  697:           jmp  sh @@EndRt
[NF2]  698: @@RtOk:   inc  dl
[E53]  699: @@EndRt:  xor  bh,bh
[6W4]  7ØØ:           VIDEO Ø2h  ; Crs um 1 nach rechts
[T31]  7Ø1:           pop bx
[3PØ]  7Ø2:           ret
[192]  7Ø3: KeyRight  endp
[ØØØ]  7Ø4:
[1B4]  7Ø5: KeyEnd    proc  ; Ende-Taste
[DV2]  7Ø6:           mov  dl,[bx.Len]
[AX2]  7Ø7:           add  dl,X_Start-1
[VA2]  7Ø8:           mov  dh,y_pos
[TC1]  7Ø9:           push bx
[LN1]  71Ø:           xor  bh,bh
[WA1]  711:           VIDEO Ø2h
[LA1]  712:           pop  bx
[3PØ]  713:           ret
[122]  714: KeyEnd    endp
```

```
[ØØØ]  715:
[XØ5]  716: KeyHome   proc          ; Home-Taste
[WW2]  717:           mov  dl,X_Start
[VA2]  718:           mov  dh,y_pos
[TC1]  719:           push bx
[LN1]  72Ø:           xor  bh,bh
[WA1]  721:           VIDEO Ø2h
[LA1]  722:           pop  bx
[3PØ]  723:           ret
[N72]  724: KeyHome   endp
[ØØØ]  725:
[ØØØ]  726:
[895]  727: ; GetKey : Auf Tastendruck warten
[497]  728: ; Ausgabe: Carry-Flag gesetzt: ESC betätigt
[YT4]  729: ;          ch = 1             : Steuertaste
[M14]  73Ø: ;    dann  cl = Ø             : Links
[MT3]  731: ;          cl = 1             : Rechts
[C33]  732: ;          cl = 2             : Up
[SM3]  733: ;          cl = 3             : Down
[893]  734: ;          cl = 4             : End
[YK3]  735: ;          cl = 5             : Home
[X63]  736: ;          cl = 1Ø            : F1Ø
[P3Ø]  737: ;
[4T4]  738: ;          ch = Ø       : Normale Taste
[K14]  739: ;          al           : ASCII- oder
[WV4]  74Ø: ;                         erweiterter Code
[ØØØ]  741:
[FY2]  742: GetKey    proc near
[TC1]  743:           push bx
[R23]  744: @@Again:  xor  ah,ah
[Z61]  745:           int  16h
[BY4]  746:           ; Steuertaste al=Ø,ah=erw. Code
[CC1]  747:           cmp  al,Ø
[3E5]  748:           ; Je Steuertaste, desto hüpf...
[522]  749:           jz   @@CrsKey
[RN2]  75Ø:           ; normale Tasten
[RW4]  751:           cmp  al,_ESC   ; ESC betätigt?
[DF1]  752:           jnz  @@CR
[HD5]  753:           ; Richtungs-Flag setzen bei ESC
[ENØ]  754:           stc
[4C3]  755:           jmp  sh @@EndGetKey
[2F5]  756: @@CR:     cmp  al,CR     ; CR betätigt?
[6V4]  757:           jnz  sh @@LF   ; Nein -> Weiter
[W85]  758:           jmp  sh @@Failure ; Ja   -> Fehler
[MN5]  759: @@LF:     cmp  al,LF       ; LF betätigt?
[UØ2]  76Ø:           jnz  sh @@BS
[LW2]  761:           jmp  sh @@Failure
[ZA5]  762: @@BS:     cmp  al,BS       ; Backspace?
[HH5]  763:           jnz  sh @@Weiter  ; Nein -> Weiter
[W85]  764:           jmp  sh @@Failure ; Ja   -> Fehler
[9K1]  765: @@Weiter:
[SN1]  766:           xor  ch,ch
[125]  767:           jmp  sh @@EndGetKey ; und Ende!
[4F2]  768:           ; Steuertasten
[6G1]  769: @@CrsKey:
[R63]  77Ø:           cmp  ah,F1Ø        ; F1Ø?
[332]  771:           jnz  sh @@Up
[PK1]  772:           mov  cl,1Ø
[222]  773:           jmp  @@CrsOk
[AH4]  774: @@Up:     cmp  ah,CrsUp      ; Up?
[PD2]  775:           jnz  sh @@Down
[8E1]  776:           mov  cl,2
[222]  777:           jmp  @@CrsOk
[N45]  778: @@Down:   cmp  ah,CrsDn      ; Down?
[WD2]  779:           jnz  sh @@Left
[AE1]  78Ø:           mov  cl,3
[222]  781:           jmp  @@CrsOk
[N95]  782: @@Left:   cmp  ah,CrsLeft    ; Left?
[6S2]  783:           jnz  sh @@Right
[MP1]  784:           xor  cl,cl
[222]  785:           jmp  @@CrsOk
[XM5]  786: @@Right:  cmp  ah,CrsRight   ; Right?
[8T2]  787:           jnz  sh @@EndPos
[6E1]  788:           mov  cl,1
[222]  789:           jmp  @@CrsOk
[S85]  79Ø: @@EndPos: cmp ah,EndPos      ; End?
[RC2]  791:           jnz  sh @@Home
[CE1]  792:           mov  cl,4
[222]  793:           jmp  @@CrsOk
[DX4]  794: @@Home:   cmp  ah,Home       ; Home?
[YX2]  795:           jnz  sh @@Failure
[EE1]  796:           mov  cl,5
[916]  797: @@CrsOk:  ; Flag "Steuertaste" setzen
[7V1]  798:           mov  ch,Ø1h
[S44]  799:           xchg al,ah         ; Code in al!
[4C3]  8ØØ:           jmp  sh @@EndGetKey
[DR1]  8Ø1: @@Failure:
[DD3]  8Ø2:           mov  dl,Ø7         ; Beep!
[161]  8Ø3:           DOS  Ø2h
[ZY1]  8Ø4:           jmp  @@Again
[A32]  8Ø5: @@EndGetKey:
[LA1]  8Ø6:           pop  bx
[3PØ]  8Ø7:           ret
[S22]  8Ø8: GetKey    endp
[ØØØ]  8Ø9:
[GW4]  81Ø: Code      Ends               ; Das war's!!!
[4K1]  811:           End Start
```

Listing 2. Mit »cfgedit.asm« verändern Sie die »config.sys« während des Bootens



FDC-Werkstatt

Gut in der Spur

Der Floppy Disk Controller (Diskettencontroller) oder kurz »FDC« bildet im PC die Schnittstelle zwischen Computer und Diskettenlaufwerk. Sie erfahren im folgenden, wie Sie diese interessante Hardwarekomponente über Ein- und Ausgabeports ohne BIOS-Funktionen programmieren.

Normalerweise programmieren Sie ein Diskettenlaufwerk über den BIOS-Interrupt 13hex, mit dem Sie alle Sektoren einer Diskette gezielt lesen, beschreiben oder überprüfen. Die Funktion 5 des Interrupts 13hex gestattet Ihnen zusätzlich, Spuren einer Diskette zu formatieren. Diese Art der Programmierung birgt jedoch einen großen Nachteil in sich: Der Diskettenin-

terrupt 13hex stellt nur die elementaren Funktionen des FDC-PD765 zur Verfügung. Eine weitaus bessere Alternative besteht darin, den FDC direkt über Ein- und Ausgabeports zu programmieren.

Um den Diskettencontroller direkt zu programmieren, benötigen Sie noch zusätzliche Informationen über die Organisation einer Spur. Die Tatsache, daß eine Spur 9 bis 18 Sektoren aufnimmt, ist Ihnen bekannt und gibt dennoch nicht die Gesamtstruktur einer Spur wieder.

Der FDC legt bei der Formatierung einer Spur noch weitere Strukturen an, die Sie im folgenden als »interne Bereiche« kennenlernen werden, da Sie keinen direkten Zugriff auf diese Bereiche besitzen. Die internen Bereiche benötigt der FDC, um zum Beispiel zu überprüfen, ob er einen gelesenen

Register	Port	Zugriff
Digitales Ausgaberegister	3F2hex	Schreiben
Hauptstatusregister	3F4hex	Lesen
Kommandoregister	3F5hex	Schreiben
Statusregister	3F5hex	Lesen
Digitales Eingaberegister (nur AT)	3F7hex	Lesen
Diskettenkontrollregister (nur AT)	3F7hex	Schreiben

Tabelle 1. Die Portadressen des FDC

oder geschriebenen Sektor fehlerfrei übertragen hat. Sehen Sie sich hierzu das Bild an, das Sie über den detaillierten Aufbau einer Spur informiert. Die Abbildung und alle folgenden Beschreibungen gehen davon aus, daß sich neun Sektoren auf einer Spur befinden, da diese Anzahl für PCs allgemeingültig ist.

Das Wort »gap« bedeutet übersetzt »Lücke«. Allen Gaps ist gemein, daß sich ihr Inhalt aus dem Wert 4Ehex zusammensetzt. Gap 4A leitet eine Spur ein. Sie ist 80 Byte lang und gleicht Toleranzen aus, die bei Schreib- oder Lesevorgängen auftreten. Bei einem Zugriff auf eine Spur ist es von der heutigen Technologie her nicht machbar, die Schreib-/Leseköpfe des Diskettenlaufwerks direkt an den Anfang einer Spur zu positionieren. Um jedoch sicherzustellen, daß der Diskettencontroller die nachfolgenden wichtigen Bereiche nicht überliest, haben die Konstrukteure des FDC-PD765 die Gap 4A am Anfang jeder Spur vorgesehen.

Der FDC überliest beim Zugriff auf eine Spur zunächst alle 4Ehex-Bytes, die er am Anfang der Spur findet, und gelangt so hundertprozentig zu dem nächsten Bereich.

Auf die Gap 4A folgt ein Sync-Bereich. Dieser Bereich wiederholt sich mehrmals innerhalb einer Spur und ist immer aus 12 Nullbyte aufgebaut. Das Kürzel »Sync« steht hier für Synchronisation und stellt eine Notwendigkeit für die Laufwerkselektronik, also für das im PC verwendete MFM-Aufzeichnungsverfahren dar. Diese Vorgänge sind jedoch rein physikalischer Natur, weshalb sie im folgenden nicht weiter betrachtet werden. Nach dem Sync-Bereich sehen Sie zum ersten Mal eine »Adress Mark« (Adressenmarkierung), und zwar die Index Adress Mark »IAM«. Alle Adressenmarkierungen setzen sich aus 3 Byte mit dem Wert A1hex zusammen. Die Index Adress Mark kennzeichnet für den FDC den eigentlichen Beginn der Spur. Sie können sie sich als eine Art »Aufmerksamkeitserreger« vorstellen, der dem Diskettencontroller anzeigt, daß nun die wichtigen Bereiche der Spur folgen. Auf die IAM folgt ein Byte mit dem Wert FChex. Die Aufgabe dieses Bytes ist nicht ganz eindeutig.

Da jedoch immer auf eine Adressenmarke ein einzelnes Byte folgt, können Sie davon ausgehen, daß dieses Byte dem FDC anzeigt, um welche Adressenmarke es sich handelt. Dieser Interpretation zufolge kennzeichnet der Wert FChex, daß die vorangehende Adressenmarke die Index Adress Mark darstellt. Um sie zu verarbeiten, benötigt der Diskettencontroller eine kurze Zeiteinheit. Um dem FDC diese Zeit zur Verfügung zu stellen, folgt eine weitere Lücke (Gap 1). Gap 1 ist notwendig, da sich die Diskette im Laufwerk permanent weiterdreht und der FDC durch das Fehlen dieser Lücke Teile der folgenden Bereiche ignorieren würde. Die beschriebenen Bereiche treten nur einmal zu Beginn einer Spur auf und stellen gewissermaßen den Kopf der Spur dar. Alle nun folgenden Bereiche (bis auf Gap 4B) wiederholen sich für jeden Sektor der Spur.

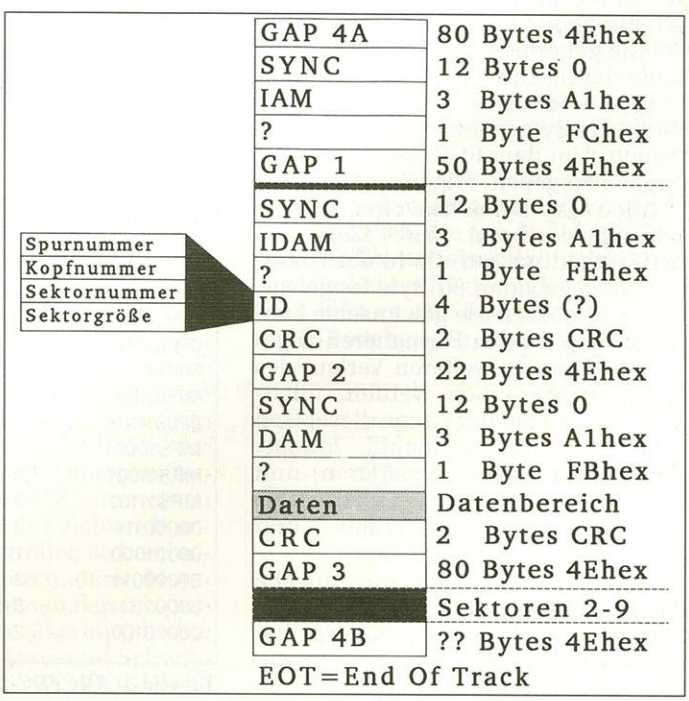
Alle Sektoren weisen komplizierte Strukturen auf

Jeder Sektor beginnt mit einem Sync-Bereich, auf den die ID Adress Mark »IDAM« folgt. Sie teilt dem FDC den Beginn eines Sektors mit. Das auf die IDAM folgende Byte FEhex identifiziert die vorangegangene Adressenmarke als IDAM. Die nächsten vier Byte, die man als Sektor-ID oder nur ID bezeichnet, enthalten für den Sektor spezifische Informationen. Das Kürzel »ID« bedeutet Identifizierung (englisch »identification«). Stellen Sie sich die ID als Visitenkarte eines Sektors vor. Der Diskettencontroller differenziert anhand der Sektor-ID zwischen den einzelnen Sektoren einer Spur. Die Sektor-ID setzt sich aus folgenden Komponenten zusammen: Spurnummer, Kopfnummer, Sektornummer und Sektorgröße. Wenn Sie sich schon des Disketteninterrupts 13hex bedient haben, kennen Sie diese Angaben von der Formatierroutine (Funktion 5). Dort müssen Sie vor dem Aufruf der Funktion im Registerpaar »ES:BX« einen Zeiger auf einen Bereich übergeben, der für jeden Sektor der zu formatierenden Spur diese vier Angaben beinhaltet. Die Spurnummer darf je nach Laufwerktyp zwischen 0 und 39 (40 Spuren) beziehungsweise zwischen 0 und 79 (80 Spuren) liegen. Die Kopfnummer liegt zwischen 0 und 1 und die Sektornummer zwischen 1 und 9, bei höheren Schreibdichten zwischen 1 und 15 (1,2 MByte) oder 18 (1,44 MByte). Die Sektorgröße liegt als verschlüsselter Wert vor. Sie erkennen dies bereits an der Tatsache, daß Sie die Sektorgröße in nur einem Byte angeben. Die Größenangabe von 512 Byte benötigt aber 2 Byte oder ein Wort.

Wenn ein Sektor (wie im MS-DOS-Format) 512 Byte Daten aufnimmt, ist dort der Wert 2 verzeichnet. Die effektive Länge eines Sektors berechnen Sie mit der Formel:

$$2^{n+7}$$

wobei n der eben angesprochene Wert ist. Für »n = 2« ergibt dies:



Der detaillierte Aufbau einer Spur

Bit	Belegung (Bit gesetzt)
7	Motor Laufwerk D einschalten
6	Motor Laufwerk C einschalten
5	Motor Laufwerk B einschalten
4	Motor Laufwerk A einschalten
3	Interrupt und DMA unterstützen
2	Reset des FDC
1,0	Laufwerk selektieren (0 bis 3)

Tabelle 2. Das digitale Eingaberegister

2^9=512.

Auf die Sektor-ID folgt ein 2 Byte großes Feld, das eine Prüfsumme enthält. Diese berechnet der Diskettencontroller über die ID Adress Mark, das Byte FEhex und die Sektor-ID, um deren Richtigkeit zu überprüfen. Der FDC bildet die Prüfsumme nach dem CRC-Verfahren (»Cyclic Redundancy Check«), das für Sie nicht von Bedeutung ist, da diese Vorgänge intern ablaufen. Hinter dem CRC-Feld befindet sich Gap 2, die dem FDC genügend Zeit gibt, um die vorangegangenen Bereiche zu verarbeiten, die CRC-Prüfsumme zu bilden und diese zu überprüfen. Nach einem weiteren Sync-Bereich folgt die Data Adress Mark »DAM«, die in Interaktion mit dem FBhex-Byte dem FDC den eigentlichen Datenbereich eines Sektors ankündigt. Die Länge des Datenbereichs hängt von der vierten Angabe der Sektor-ID ab; beim Standard-MS-DOS-Format umfaßt dieser Bereich 512 Byte.

Gap 3 steigert die Datensicherheit des Aufzeichnungsformats

Die Überprüfung der nachfolgenden CRC-Prüfsumme ist ein Garant dafür, daß die gelesenen oder geschriebenen Daten richtig sind. Der FDC bildet diese Prüfsumme über die Data Adress Mark, das Byte FBhex und den Datenbereich. Um dem FDC Zeit für diesen Vorgang zu geben, folgt Gap 3, die unter allen Gap-Bereichen eine Sonderstellung einnimmt. Ihre Länge ist nicht unbedingt auf die in der Abbildung angegebenen 80 Byte festgelegt, denn Sie müssen die gewünschte Länge von Gap 3 beim Formatieren angeben. Wie Sie im weiteren Verlauf des Beitrags noch sehen werden, unterscheidet man bei der Länge dieser Gap zwischen »Gap 3 length format« (Gap-3-Länge beim Formatieren) und »Gap 3 length read/write« (Gap-3-Länge beim Lesen und Schreiben).

Gap 3 besitzt zusätzlich die Aufgabe der Toleranzausgleichung, die Sie schon bei Gap 4A kennengelernt haben. Der FDC ist nicht in der Lage, die Daten eines zu schreibenden Sektors genau in das vorgesehene Datenfeld einzutragen. Man gibt deshalb beim Formatieren einer Spur die Länge 80 für Gap 3 an, damit der FDC bei einem späteren Schreibvorgang die Bereiche des nächsten Sektors, die sich nahtlos an den vorausgehenden Sektor anschließen, nicht überschreibt. Wenn Sie die Länge von Gap 3 zu kurz wählen, überschreibt der FDC eventuell die Bereiche, die den nächsten Sektor einleiten, wodurch dessen Daten zerstört, das heißt, unlesbar werden. Die Länge von Gap 3 bei Lese- oder Schreibvorgängen ist unter MS-DOS auf 42 festgelegt, also deutlich kleiner als die Gap-3-Formatlänge. Diese Gap-3-Länge ist für den Diskettencontroller bei Schreibvorgängen von besonderem Interesse. Nachdem der FDC die von Ihrem Rechner kommenden Daten und die berechnete Prüfsumme auf das Medium geschrieben hat, erzeugt er genau wie beim Formatieren die Gap 3. Ein Beispiel verdeutlicht diese Notwendigkeit.

Sie formatieren eine Spur und versehen einen bestimmten Sektor mit Daten. Dieser erste Schreibvorgang verläuft nun sehr schlecht, das heißt, der FDC weicht beim Eintragen der Daten sehr stark von dem vorgesehenen Bereich nach hinten ab. Der Inhalt dieses Sektors überschreibt also Teile der nachfolgenden Gap 3, jedoch keine der Bereiche des nachfolgenden Sektors. Für den FDC ist dieser Sektor hervorragend geschrieben, und er kann bei Bedarf auf dessen Inhalt wieder zurückgreifen. Ein zweiter Schreibvorgang verläuft besser, wodurch sich die Daten ziemlich genau hinter der Data Adress Mark befinden. Wenn der FDC nun bei einem Schreibvorgang keine Gap 3 erzeugt, befinden sich in diesem Beispiel hinter den neuen Daten noch Überreste der alten Daten. Der FDC erwartet jedoch hinter einem Datenbereich die Gap 3, die nur aus 4Ehex-Bytes besteht. Er muß also sicherstellen, daß die Kontinuität von Gap 3 auch nach vielen Schreibzugriffen gewahrt bleibt, und bewältigt diese Anforderung durch das Erzeugen einer Gap 3 nach jedem Schreibzugriff.

Sie müssen die Länge von Gap 3 bei allen Schreib- und Lesezugriffen angeben, wobei Sie diese Längenangabe ungefähr halb so groß wie die Gap-3-Formatlänge wählen. Die heute in PCs verwendeten Diskettenlaufwerke arbeiten bereits so präzise, daß Sie die Gap-3-Formatlänge auf 32 Byte herabsetzen dürfen, ohne daß es zu Problemen kommt. Der so gewonnene Platz auf dem Speichermedium läßt dann sogar zu, daß Sie bei Double-Density-Laufwerken zehn statt neun Sektoren pro Spur formatieren können. Sie erhöhen dadurch die Speicherkapazität einer DD-Diskette auf 400 KByte.

Eines von 15 Kommandos setzen Sie an den FDC zur Bearbeitung ab

Nachdem der FDC die Bereiche aller Sektoren beim Formatieren angelegt hat, bleiben in der Regel noch einige Bytes frei, welche die Gap 4B bilden. Die Bezeichnung »EOT« in der Abbildung steht für »End Of Track« (Ende der Spur). Es handelt sich hierbei nicht um einen Bereich, sondern vielmehr um ein Signal, das der FDC zum Beispiel sendet, wenn Sie auf einen Sektor zugreifen, der überhaupt nicht existiert.

Die Programmierung des FDC-PD765 verläuft über Ein- und Ausgabeports, deren Adressen Sie in Tabelle 1 sehen. Mit Hilfe des digitalen Ausgaberegisters steuern Sie die Motoren aller angeschlossenen Laufwerke, stellen besondere Betriebsarten des FDC ein und selektieren das Laufwerk, das für Sie eine Diskettenoperation durchführt. Tabelle 2 zeigt die Verwendung der einzelnen Bits des digitalen Aus-

Befehlscode	Kommandophase	Ergebnisphase	Funktion
MFS00110	9 Byte	7 Byte	Sektor lesen
0FS00010	9 Byte	7 Byte	Spur lesen
MFS01100	9 Byte	7 Byte	gelöschten Sektor lesen
0F001010	2 Byte	7 Byte	Sektor-ID lesen
MF000101	9 Byte	7 Byte	Sektor schreiben
MF001001	9 Byte	7 Byte	gelöschten Sektor schreiben
0F001101	6 Byte	7 Byte	Spur formatieren
MFS10001	9 Byte	7 Byte	Sektor testen (gleich)
MFS11001	9 Byte	7 Byte	Sektor testen (kleiner-gleich)
MFS11101	9 Byte	7 Byte	Sektor testen (größer-gleich)
00000111	2 Byte	0 Byte	Rekalibrieren
00001000	1 Byte	2 Byte	Interruptstatus ermitteln
00000011	3 Byte	0 Byte	Laufwerkdaten festlegen
00001111	3 Byte	0 Byte	Spur anfahren
00000100	2 Byte	1 Byte	Laufwerkstatus ermitteln

Tabelle 3. Alle FDC-Befehlscodes im Überblick

Byte	Bedeutung
1	Befehlscode
2	Laufwerk und Kopf: Bits Belegung 3-7 nicht benutzt 2 Kopf (0 oder 1) 0,1 Laufwerk (0 bis 3)
3	Spur
4	Kopf
5	Sektor
6	Sektorgröße
7	letzter Sektor
8	Gap-3-Länge (»Read/Write«)
9	Datenlänge (weniger 1)

Tabelle 4. So sieht die Kommandophase für alle neun Bytekommandos aus

gaberegisters. Die oberen vier Bits (4 bis 7) steuern die Zustände der Laufwerksmotoren. Durch Manipulieren eines oder mehrerer Bits schalten Sie die Motoren bestimmter Laufwerke ein (Bit setzen) oder aus (Bit löschen). Das dritte Bit benutzen Sie, um den DMA-Controller Ihres Rechners mit allen Datenübertragungen von und zum Diskettenlaufwerk zu beauftragen (Bit setzen). Der FDC löst darüber hinaus mehrmals während der Abarbeitung eines Befehls eine Interruptanforderung (IRQ6) aus, um die Aufmerksamkeit des Kontrollprogramms zu erlangen. Diese Interruptanforderung führt dazu, daß der Interruptcontroller die Routine des Interrupts 0Ehex auslöst. Dort befindet sich eine Routine, die das achte Bit der Speicherstelle 0040:003Fhex setzt. Das BIOS fragt den Zustand dieses Bits ab, um zum Beispiel in eine weitere Befehlsphase überzugehen. Durch Löschen des vierten Bits im digitalen Ausgaberegister sperren Sie die entsprechenden Treiber. Durch Setzen des dritten Bits führen Sie einen Reset des FDC und aller angeschlossenen Laufwerke durch. Setzen Sie dieses Bit bei jedem Manipulationsvorgang des digitalen Ausgaberegisters auf 1. In die Bits 0 und 1 schreiben Sie die Nummer des Diskettenlaufwerks, das Sie anwählen (0 für A, 1 für B etc.).

Ein Kommando für den FDC umfaßt 9 Byte der Kommandophase

Sehen Sie sich Tabelle 3 an. Sie zeigt Ihnen alle verfügbaren FDC-Kommandos. In der linken Spalte der Tabelle sehen Sie die FDC-Befehlscodes in binärer Schreibweise. Die Bits »M«, »F« und »S« sind hierbei variabel. Durch das Setzen oder Löschen dieser Spezialbits stellen Sie für einen FDC-Befehl Optionen ein, die der FDC bei der Ausführung berücksichtigt. Das M-Bit steht für »Multi Track«. Wenn Sie dieses Bit im Befehlscode setzen, fährt der FDC mit der Ausführung des entsprechenden Befehls auf der zweiten Diskettenseite fort, sobald er auf der ersten Seite das Ende der Spur erreicht hat. Das F-Bit (»Frequency Modulation«) stellt das Aufzeichnungsverfahren des FDC ein. Da im PC das MFM-Aufzeichnungsverfahren Verwendung findet, setzen Sie dieses Bit immer auf 1. Wie das S-Bit verwendet wird, erfahren Sie später.

Die Ausführung eines FDC-Kommandos erfolgt immer in drei Phasen: Kommando-, Ausführungs- und Ergebnisphase. In der Kommandophase übermitteln Sie dem FDC durch das Kommandoregister eine bestimmte Anzahl Bytes, die dieser für die Ausführung des entsprechenden Befehls benötigt. Die Anzahl der zu übermittelnden Bytes entnehmen Sie der zweiten Spalte (Tabelle 3). Das erste Byte der Kommandophase ist hierbei immer der Befehlscode. Die Ausführungsphase läuft intern ab. Sie dient unter anderem dazu, Daten vom Hauptspeicher zum Speichermedium oder umgekehrt zu übertragen. Dieser Arbeit nehmen sich der FDC- und der DMA-Controller gemeinsam an. Sie müssen hierzu den DMA-Controller vor Beginn der Kommandophase programmieren. In der Ergebnisphase übermittelt Ihnen der FDC eine bestimmte Anzahl Bytes (dritte Spalte in Tabelle 3), die Sie über Erfolg oder Mißerfolg der vorangegangenen Ausführungsphase informieren. Die Ergebniswerte erscheinen im Statusregister.

Richten Sie zunächst Ihr Augenmerk auf die Kommandos, deren Kommandophase 9 Byte umfassen. Tabelle 4 zeigt Ihnen die für diese Befehle erforderlichen Inhalte der 9 Kommandobytes auf. Als erstes Byte übermitteln Sie dem FDC den Befehlscode. Das zweite Byte setzt sich aus der Laufwerknummer (Bits 0 und 1) und der Kopfnummer zusammen. Als Laufwerknummer geben Sie den gleichen Wert an, den Sie in die unteren beiden Bits des digitalen Ausgaberegisters schreiben. Für die Kommandobytes 3 bis 6 müssen Sie das bereits erworbene Wissen über den Aufbau einer Spur anwenden. Diese Bytes spiegeln die Werte der Sektor-ID eines Sektors wider, den Sie bearbeiten. Das siebte Kommandobyte dient dazu, mehrere Sektoren durch ein FDC-Kommando zu bearbeiten. Wenn die Werte der Kommandobytes 5 und 7 übereinstimmen, bearbeitet der FDC nur einen Sektor. Ist der Wert des siebten Kommandobytes jedoch größer, so bearbeitet der FDC alle Sektoren, deren Nummern in den angegebenen Bereich passen. Als achtes Byte übermitteln Sie die Gap-3-Länge für Lese- und Schreiboperationen, die unter MS-DOS bei 42 liegt. Als Datenlänge (Kommandobyte 9) übergeben Sie den Wert -1.

Bei den drei Testkommandos besitzt das neunte Byte eine andere Bedeutung. Bei diesen Kommandos übermitteln Sie als neuntes Byte einen Wert zwischen 1 und 2. Dieser Wert stellt die Verhaltensweise des FDC bei einem Testvorgang ein. Die genaue Funktion ist nicht bekannt und konnte auch nicht ermittelt werden. Der BIOS-Diskette-ninterrupt 13hex stellt als Funktion 4 einen Weg zur Verfügung, Sektoren einer Diskette zu verifizieren. Sie benutzt hierbei jedoch nicht die Testfunktion »auf Gleichheit testen«, sondern bedient sich des FDC-Kommandos »Sektor lesen«. Den Vergleich der Daten übernimmt der DMA-Controller. Er vergleicht jedes vom FDC kommende Byte mit einem Byte des Hauptspeichers. Sobald die DMA ein ungleiches Byte findet, bricht sie die FDC-Operation ab. Mit Hilfe der Ergebnisphase ermittelt das BIOS, ob der Inhalt der überprüften Sektoren mit Bereichen des Hauptspeichers übereinstimmt. Sie benötigen die drei Testkommandos also nicht unbedingt.

Gelöschte Data Adress Marks kennzeichnen versteckte Sektoren

Das FDC-Kommando »Gelöschten Sektor schreiben« schreibt einen oder mehrere Sektoren auf das Medium und löscht die Data Adress Marks der bearbeiteten Sektoren. Auf den Inhalt dieser Sektoren können Sie dann nur noch mit dem Kommando »Gelöschten Sektor lesen« zugreifen. Wenn der FDC bei einem normalen Lesekommando auf eine gelöschte DAM trifft, bricht er die Operation ab und gibt einen Fehler zurück. Durch das Setzen des S-Bits (»skip deleted Data Adress Mark«) im Befehlscode erreichen Sie, daß der FDC Sektoren mit gelöschter DAM ignoriert.

Die Ergebnisphase dieser gleichgearteten FDC-Kommandos liefert sieben Ergebnisbytes, die Sie in Tabelle 5 sehen. Die letzten vier Ergebnisbytes zeigen die Sektor-ID des zuletzt bearbeiteten Sektors. Die Bezeichnungen ST0 bis ST2 stehen für Statusregister. Ta-

Byte	Inhalt
1	ST0
2	ST1
3	ST2
4	Spur
5	Kopf
6	Sektor
7	Sektorgröße

Tabelle 5. Die Statusregister klären Sie über den Erfolg eines Kommandos auf

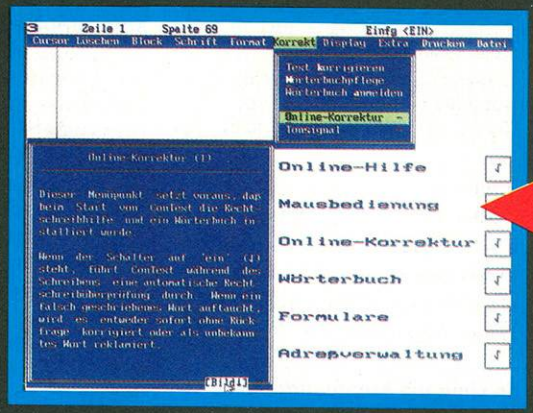
Bits	Bedeutung (wenn gesetzt)
Statusregister ST0	
6,7	Interruptcode Wert Bedeutung 0 Kommando erfolgreich beendet 1 Kommando abgebrochen 2 ungültiges Kommando 3 Kommando abgebrochen
5	»Spur anfahren« beendet
4	Fehler aufgetreten (Rekalibrieren)
3	Laufwerk nicht bereit
2	aktueller Kopf
0,1	aktuelles Laufwerk (0 bis 3)
Statusregister ST1	
7	Ende der Spur erreicht
6	nicht benutzt (immer 0)
5	Datenfehler (CRC falsch)
4	Datentransfer zwischen CPU und FDC zu langsam
3	nicht benutzt (immer 0)
2	keine Daten gefunden
1	Diskette schreibgeschützt
0	Adreßmarke nicht vorhanden
Statusregister ST2	
7	nicht benutzt (immer 0)
6	gelöschte DAM gefunden
5	CRC-Fehler im Datenbereich
4	falsche Spurnummer
3	Testkommandos: Daten gleich
2	Testkommandos: Daten ungleich
1	Falsche Spurnummer (Sektor-ID)
0	DAM oder gelöschte DAM nicht vorhanden
Statusregister ST3	
7	Fehler aufgetreten
6	Diskette schreibgeschützt
5	Laufwerk bereit
4	Köpfe befinden sich auf Spur 0
3	Doppelkopflaufwerk vorhanden
2	aktueller Kopf
0,1	aktuelles Laufwerk (0 bis 3)
Hauptstatusregister	
7	FDC ist bereit, neues Byte zu empfangen oder zu senden
6	Datenbyte für CPU verfügbar (gelöscht: FDC erwartet Byte)
5	Ausführungsphase hat begonnen
4	FDC bearbeitet Kommando
3	»Spur anfahren« oder Rekalibrieren für Laufwerk D aktiv
2	wie Bit 3, aber für Laufwerk C
1	wie Bit 3, aber für Laufwerk B
0	wie Bit 3, aber für Laufwerk A

Tabelle 6. Die Ergebnisphase umfaßt für die Befehle 7 Byte

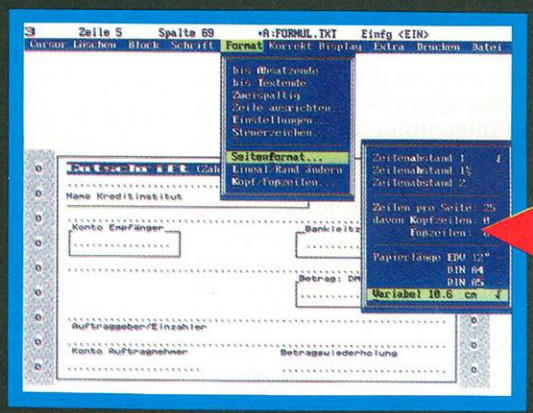
Byte	Inhalt	
1	Befehlscode (3)	
2	Schrittrate, Kopfabhebezeit	
	Bits	Belegung
	4 bis 7	Schrittrate (0=32 ms, 1=30 ms, 2=28 ms etc.)
	0 bis 3	Kopfabhebezeit (0=0 ms, 1=32 ms, 2=64 ms etc.)
3	Kopfladezeit, DMA-Modus	
	Bits	Belegung
	1 bis 7	Kopfladezeit (0=4 ms, 1=8 ms, 2=12 ms etc.)
	0	DMA-Modus (immer 0)

Tabelle 7. Das Kommando »Laufwerkdaten festlegen« erfordert 3 Kommandobytes

Die Nachricht des Jahres:



Der neuen Adreßverwaltung liegt eine dBase-kompatible Datenstruktur zugrunde, die den Datenaustausch mit externen Quellen gewährleistet. ConText 4.0 wird dadurch zusätzlich zu einem leistungsfähigen Datenverwaltungsprogramm.



Textverarbeitung mit Verstand

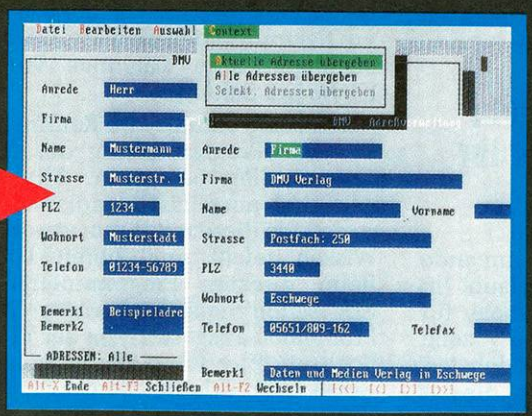
ConText ist mit mehr als 100.000 Installationen eine der erfolgreichsten Textverarbeitungen. Mit der neuen Version 4.0 wurde das ConText-Konzept überzeugend weiterentwickelt: Sie können arbeiten wie bisher, nutzen aber die Vorteile eines unschlagbaren Rechtschreibsystems, einer integrierten Adreßverwaltung und vieler Neuerungen, die das gewohnte Arbeiten noch komfortabler gestalten. Hier steckt nicht der Teufel, sondern das Genie im Detail.

ConText 4.0 – die unwiderstehliche Kombination aus Leistung und Preis.

ConText 4.0 ist da!

Kinderleicht zu bedienen – professionell einzusetzen.

Das mitgelieferte **Rechtschreiblexikon** ist lernfähig. Sie können es zu Ihrem persönlichen Duden erweitern und verändern. Während des Schreibens macht das Rechtschreibsystem Korrekturvorschläge.



UPDATE

Mit dem **Formular-Manager** ersparen Sie sich Zeit bei immer wiederkehrenden Arbeitsvorgängen. Egal ob Banküberweisung oder Formblatt – Formulare lassen sich leicht gestalten und mit dem Merkersystem problemlos ausfüllen.

CONTEXT 4.0

Das bietet ConText 4.0 Neues:

SAA-angelehnte Bedienerführung in den Menüs ■ integrierte Mausbedienung ■ Korrekturmöglichkeit während des Schreibens: Das zuschaltbare Rechtschreiblexikon ist lernfähig und erweiterbar, nicht nur um Wörter, sondern auch um ganze Sätze ■ komplett neue Adreßverwaltung: Sie importiert ASCII-Datenbestände, und ihre kompatible Datenstruktur macht Austausch und Zugang zu dBase-Daten möglich ■ in Kombination mit der Serienbrieffunktion erhält Context damit ein ausgereiftes Datenverwaltungssystem ■ der Formular-Manager erleichtert immer wiederkehrende Arbeitsvorgänge ■ der einzigartige Grafikeditor ist verbessert und erweitert worden ■ neue Undo-Funktionen ■ Löschen mit Mausbedienung ■ komfortable Dateiauswahlbox mit Wildcards ■ Kommentarzeilen für Dateien ■ automatisches Abspeichern ■ 99 Merker können im Text plaziert werden ■ zirka 300 neue Druckertreiber ■ neuer Editor zur Druckeranpassung ■ variabler Zeilenabstand (1, 1,5, 2) ■ erweiterte Tastaturbelegung mit Menüführung ■ problembezogenes Online-Hilfesystem ■ stufenlose Farbeinstellung ■ 90 Zeichen breite Textdarstellung ■ Nutzung von erweitertem Speicher

Unser Update-Service:

Sie schicken Ihre Original-Titelseite Ihres ConText-Handbuches oder Ihre Original-ConText-PRO-Disketten an untenstehende Adresse und erhalten gegen 99,– DM* die neue ConText-Version 4.0.
Bitte benutzen Sie die Update-Service-Karte im Heft.

Update **99,– DM***

ConText 4.0 **249,– DM***

ConText-Zusatzprogramm:

ConText-Zeicheneditor **99,– DM***

ASCII-Zeichensatzeditor für ConText PRO und ConText EASY

* Unabhängig von der Anzahl der bestellten Programme berechnen wir für das Inland DM 4,– bzw. für das Ausland DM 6,– Porto und Verpackung.

Bitte benutzen Sie die Bestellkarte.

DMV-Verlag • Postfach 250 • 3440 Eschwege

DMV Daten- und Medienverlag

belle 6 listet die Bits der Statusregister 0 bis 3 auf. Den Inhalt des Statusregisters ST3 liefert die Ergebnisphase des Kommandos »Laufwerkstatus ermitteln«. Das Hauptstatusregister ermitteln Sie durch einen Lesezugriff auf die Portadresse 03F4hex. Sie können das Hauptstatusregister jederzeit – auch während der Abarbeitung eines Kommandos – auslesen. Wie Sie in Tabelle 6 sehen, informiert es Sie zum Beispiel darüber, ob der FDC bereit ist, ein Byte zu empfangen (Kommandobyte) oder zu senden (Ergebnisbyte).

Die Kommandophase des Format-Kommandos umfaßt 6 Byte. Die ersten beiden Kommandobytes stimmen mit den Angaben in Tabelle 4 überein. Als drittes Byte übergeben Sie die Anzahl der Bytes pro Sektor. Als viertes Kommandobyte erwartet der FDC die Anzahl der Sektoren pro Spur. Durch das fünfte Byte spezifizieren Sie die Gap-3-Formatlänge. Der FDC füllt die einzelnen Sektoren der zu formatierenden Spur mit dem Wert, den Sie als sechstes Byte übergeben (Füllbyte). Vor der Ausführung des Formatkommandos müssen Sie den DMA-Controller so programmieren, daß sein Adreßregister auf einen Speicherbereich zeigt, in dem sich die Sektor-IDs aller Sektoren einer Spur befinden. Zum Formatieren einer Spur benötigt der FDC diese Angaben zusätzlich, die er während der Ausführungsphase aus dem Arbeitsspeicher liest.

Das Kommando »Sektor-ID lesen« benötigt eine 2 Byte lange Kommandophase. Diese beiden Bytes stimmen wiederum mit den ersten beiden Kommandobytes aus Tabelle 4 überein. In der Ausführungsphase liest der FDC die Sektor-ID des Sektors, der sich gerade unter dem Schreib-/Lesekopf befindet. Die Ergebnisphase für diesen Befehl sehen Sie in Tabelle 6. Die ersten vier Ergebnisbytes stellen die gelesene Sektor-ID dar. Mit Hilfe dieses Kommandos können Sie auf einfache Weise die Speicherkapazität einer eingelegten Diskette bestimmen. Sie lesen hierzu 18mal hintereinander die Sektor-IDs einer Spur. Die größte gefundene Sektornummer spiegelt dann die Anzahl der Sektoren pro Spur wider.

Durch das Kommando »Spur anfahren« positionieren Sie die Schreib-/Leseköpfe eines Laufwerks über einer Spur. Die ersten beiden Kommandobytes stimmen auch bei diesem Kommando mit den Angaben überein, die Ihnen Tabelle 4 vermittelt. Als drittes Byte geben Sie die gewünschte Spurnummer an. Dieses Kommando liefert keine Ergebnisse. Der FDC-Befehl »Rekalibrieren« arbeitet ähnlich wie das vorherige Kommando. Er positioniert die Schreib-/Leseköpfe auf Spur 0. Die Kommandophase umfaßt im Gegensatz zum Kommando »Spur anfahren« nur 2 Byte, da die Spurnummer immer 0 ist. Das BIOS benutzt dieses Kommando nach jedem aufgetretenen Diskettenfehler.

Der Befehl »Interruptstatus ermitteln« liefert Ihnen in der Ergebnisphase 2 Byte. Das erste Byte ist der Inhalt des Statusregisters ST0 der letzten FDC-Operation. Als zweites Byte übergibt Ihnen der FDC die aktuelle Spurnummer, auf der sich die Köpfe des Laufwerks befinden. Den FDC-Befehl »Laufwerkstatus ermitteln« haben Sie bereits kennengelernt. Die Kommandophase setzt sich auch hier aus den ersten beiden Bytes von Tabelle 4 zusammen. Die Ergebnisphase liefert Ihnen den Inhalt des ST3-Registers.

Die Schrittrate: Geschwindigkeitsbegrenzung für Ihr Laufwerk

Das noch verbleibende Kommando »Laufwerkdaten festlegen« erlaubt Ihnen, bestimmte Betriebsparameter für den FDC anzugeben. Tabelle 7 zeigt die erforderlichen Kommandobytes. Die Schrittrate gibt die Zeit an, die der FDC zwischen zwei Schrittimpulsen wartet. Die Erklärung dieser Vorgänge geht weit in die Physik. Vereinfachend gesagt: Je kleiner Sie die Schrittratenzeit spezifizieren, desto schneller arbeitet das Laufwerk. Sie können also mit diesem Wert die Geschwindigkeit des Laufwerks einstellen. Bei der Angabe der Schrittrate dürfen Sie einen bestimmten Schwellenwert nicht unterschreiten, da sonst zu viele Schreib- und Lesefehler auftreten, die zur Beschädigung Ihres Laufwerks führen. Der Wert »Head Load Time« (Kopfladezeit) gibt die Zeit an, die der FDC wartet, bevor er die Köpfe auf das Medium herunterläßt. Der Wert »Head Unload Time« gibt eine Verzögerungszeit an, die der FDC nach einer Operation wartet, bevor er die Köpfe wieder vom Medium entfernt. Für alle drei Parameter gilt, daß diese im BIOS aus Datensicherheitsgründen höher als benötigt angesetzt sind. Durch Optimieren dieser Werte können Sie Ihr Laufwerk bis zu einem bestimmten Punkt beschleunigen. Das DMA-Modus-Bit müssen Sie immer auf 0 setzen.

Wenn Sie einen AT besitzen, stehen Ihnen zusätzliche Funktionen zur Verfügung. Durch einen Lesezugriff auf das digitale Eingaberegister stellen Sie fest, ob sich eine Diskette im aktuell angewählten Diskettenlaufwerk befindet. Ein gesetztes achtes Bit zeigt Ihnen an, daß Sie die Klappe des Laufwerks seit dem letzten FDC-Kommando geöffnet haben. Tabelle 8 listet die Benutzung der Bits des Diskettenkontrollregisters auf. Sie erreichen dieses Register durch einen Schreibzugriff auf Port 03F7hex. In diesem Register finden nur die unteren beiden Bits Verwendung. Die Datenrate für Double-Density-Laufwerke liegt bei 250 KByte pro Sekunde. High-Density-Laufwerke benötigen eine Datenrate von 500 KByte pro Sekunde.

Das Listing enthält alle Routinen, die Sie zum Programmieren des Diskettencontrollers benötigen. Alle Routinen sind ausreichend dokumentiert, so daß Sie diese mit Ihrem erworbenen Wissen leicht analysieren können. Beim direkten Programmieren des FDC sehen Sie sich enormen Timing-Problemen ausgesetzt, da der FDC ein hochsensibler IC ist. Es ist beispielsweise nicht erlaubt, dem FDC in der Kommandophase einfach alle erforderlichen Werte hintereinander zu senden. Sie müssen zwischen jedem zu sendenden Byte den Zustand des FDC durch Auslesen des Hauptstatusregisters ermitteln. Sofern der FDC bereit ist, ein Byte zu empfangen, können Sie ihm dieses übermitteln. Die Routinen des Listings sind bereits in der Praxis erprobt, so daß Sie diese ohne Modifizierungen in eigene Programme übernehmen können. Alle Routinen nehmen Ihnen zudem sämtliche Timing-Probleme ab.

Die Beispielsequenz des Listings liest den Bootsektor der Diskette, die sich beim Starten des Programms in Laufwerk A befindet. Das Programm liest zunächst den Bootsektor durch den Interrupt 25hex, den auch MS-DOS benutzt. Danach erfolgt das Lesen des Sektors durch direkten FDC-Zugriff. Laden Sie den MS-DOS-Debugger »debug«:

```
debug fdc.com
```

und starten Sie das Programm durch Eingabe des Buchstabens »g«. Danach

Bits	Belegung	
2-7	nicht benutzt	
0-1	Datenübertragungsrate	
	Wert	Datenrate
	0	500 KByte pro Sekunde
	1	300 KByte pro Sekunde
	2	250 KByte pro Sekunde
	3	1 MByte pro Sekunde (nicht immer unterstützt)

Tabelle 8. Die Datenübertragungsrate stellen Sie über das Diskettenkontrollregister ein

listen Sie den Speicherbereich ab Adresse 048Dhex mit dem Dump-Kommando »d« auf. Ab dieser Adresse muß dann zweimal hintereinander der Bootsektor stehen.

Das Programm benutzt den DMA-Controller, um den Sektor in den Hauptspeicher Ihres Rechners zu übertragen. In der DOS International, Ausgabe 3'91 befindet sich eine DMA-Werkstatt, die Ihnen die Grundlagen vermittelt. Im Datenbereich des Listings befinden sich einige Speicherstellen, die die für das Programm notwendigen Wartezeiten enthalten. Bei Benutzung eines sehr schnellen Rechners müssen Sie diese Wartezeiten erhöhen.

(Ingo Kubbilun/db)

Literatur:
Jörg Koch: PC/XT/AT-Kompendium, Markt&Technik Verlag, 500 Seiten, ISBN 3-89090-778-4, 69 Mark

Rolf Brückmann, Jörg Schieb: Das große CPC-Floppybuch, Data Becker Verlag, 350 Seiten, ISBN 3-89011-093-2, 49 Mark

Intel: Model 401 Board Technical Reference Manual, Intel Corporation, Bestellnummer 504366-002

Amstrad: Amstrad PC Technical Reference Manual, 300 Seiten, Amstrad Consumer Electronics Plc, 89 Mark

```
[RX2]    1: ;Programm: fdc.asm
[ØR6]    2: ;Funktion: Bereitstellung von Routinen
[BA6]    3: ;          zur direkten FDC-Programmierung
[AL5]    4: ;Sprache : MASM ab Version 4.Ø, TASM
[GW3]    5: ;Autor   : Ingo Kubbilun
[SH5]    6: ;          (c)1991 DMV Widuch GmbH & Co. KG
[ØØØ]    7:
[FB3]    8: CODE       segment byte
[LY3]    9:            assume  CS:CODE, DS:CODE
[4Ø4]   1Ø:            assume  ES:CODE, SS:CODE
[ØØØ]   11:
[3M1]   12:            org     Ø1ØØh
[ØØØ]   13:
[K53]   14: start:     jmp     init
[ØØØ]   15:
[AT4]   16: ; Daten- und Variablenbereich
[ZØ2]   17: dmacom     db ?
[ZU4]   18:   ;Kommando für DMA-Kontroller
[3C2]   19: oldintØ8   dw ?,?
[GC2]   2Ø: oldintØe   dw ?,?
[S92]   21: counter    db ?
[HB2]   22: fdcstatus  db ?
[PC2]   23: montime    db 32h
[CF5]   24:   ;Wartezeit nach Einschalten des
[9Z3]   25:   ;Motors in 1/1ØØstel sec.
[5E2]   26: mofftime   db 64h
[S45]   27:   ;Wartezeit: Motor ausschalten
[DB2]   28: fdcdelay   dw 2Øh
[F26]   29:   ;Verzögerungszeit für FDC-Zugriff
[D72]   3Ø: intflag    db ?
[2B2]   31: motwert    db ?
[7H2]   32: rekalretry dw 4
[KØ6]   33:   ;Anzahl Wiederholungsversuche beim
[KR4]   34:   ;Rekalibrieren des Laufwerks
[PC2]   35: headsetdel db 4
[EE7]   36:   ;Verzögerungszeit für ¤Spur anfahren'
[TY3]   37:   ;und ¤Rekalibrieren'
[893]   38: result     db 8 dup (?)
[SS3]   39:   ;Statusinformationen
[N13]   4Ø: writedelay dw Ø4ØØh
[K75]   41:   ;Wartezeit nach Schreibzugriff
[VN2]   42: resuldelay dw 1Øh
[EV4]   43:   ;Wartezeit beim Auslesen der
[SS3]   44:   ;Statusinformationen
[FF2]   45: exeretry   db Ø8h
[1Ø5]   46:   ;Wiederholungsversuche beim
[EC5]   47:   ;Übergang in Ausführungsphase
[6V1]   48: drhd       db ?
[BØ2]   49: retry      db ?
[ØØØ]   5Ø:
[6L2]   51: ;Hilfsroutinen
[4E3]   52: fdcinit    proc    near
[Ø14]   53:   ;Interrupts 8 und ØEhex
[RL4]   54:   ;patchen und Timer auf 1ØØ mal
[UE3]   55:   ;pro Sekunde setzen
[2H2]   56:            cmp     intflag,Ø
[NZ1]   57:            jz      initok
[3PØ]   58:            ret
[314]   59: initok:    mov     intflag,-1
[EMØ]   6Ø:            cli
[ULØ]   61:            cld
[MF1]   62:            push    DS
[PF1]   63:            push    ES
[GV1]   64:            xor     AX,AX
[3V1]   65:            mov     ES,AX
[WX1]   66:            mov     DI,2Øh
[YT2]   67:            lds     BX,ES:[DI]
[VC3]   68:            mov     CS:oldintØ8,BX
[TK3]   69:            mov     CS:oldintØ8+2,DS
[T24]   7Ø:            mov     AX,offset newintØ8
[ZYØ]   71:            stosw
[2V1]   72:            mov     AX,CS
[ZYØ]   73:            stosw
[DY1]   74:            mov     DI,38H
[YT2]   75:            lds     BX,ES:[DI]
[FE3]   76:            mov     CS:oldintØe,BX
[DM3]   77:            mov     CS:oldintØe+2,DS
[Ø44]   78:            mov     AX,offset newintØe
[ZYØ]   79:            stosw
[2V1]   8Ø:            mov     AX,CS
[ZYØ]   81:            stosw
[GD1]   82:            pop     ES
[ED1]   83:            pop     DS
[CY1]   84:            mov     AL,36h
[ØØ2]   85:            out     43h,AL
[CZ1]   86:            mov     AL,9Ch
[RZ1]   87:            out     4Øh,AL
[9Z1]   88:            mov     AL,2Eh
[RZ1]   89:            out     4Øh,AL
[6PØ]   9Ø:            sti
[3PØ]   91:            ret
[1D2]   92: fdcinit    endp
[ØØØ]   93:
[VP3]   94: fdcrestore proc    near
[DP4]   95:   ;Interrupts Ø8hex und ØEhex
[ZV2]   96:   ;reinstallieren
[2H2]   97:            cmp     intflag,Ø
[GE2]   98:            jnz     deintok
[3PØ]   99:            ret
[DØ4]  1ØØ: deintok:   mov     intflag,Ø
[EMØ]  1Ø1:            cli
[ULØ]  1Ø2:            cld
[PF1]  1Ø3:            push    ES
[GV1]  1Ø4:            xor     AX,AX
[3V1]  1Ø5:            mov     ES,AX
[WX1]  1Ø6:            mov     di,2Øh
[PX2]  1Ø7:            mov     AX,oldintØ8
[ZYØ]  1Ø8:            stosw
[523]  1Ø9:            mov     AX,oldintØ8+2
[ZYØ]  11Ø:            stosw
[DY1]  111:            mov     DI,38h
[AZ2]  112:            mov     AX,oldintØe
[ZYØ]  113:            stosw
[T33]  114:            mov     AX,oldintØe+2
[ZYØ]  115:            stosw
[CY1]  116:            mov     AL,36h
[ØØ2]  117:            out     43h,AL
[MT1]  118:            xor     AL,AL
[RZ1]  119:            out     4Øh,AL
[RZ1]  12Ø:            out     4Øh,AL
[GD1]  121:            pop     ES
[6PØ]  122:            sti
[3PØ]  123:            ret
[SN2]  124: fdcrestore endp
[ØØØ]  125:
[ZH3]  126: motoron    proc    near
[6Ø5]  127:   ;Laufwerksmotor einschalten
[XK3]  128:   ;DL=Nummer (Ø=A, 1=B,...)
[EMØ]  129:            cli
[BH1]  13Ø:            and     DL,3
[NT1]  131:            mov     CL,DL
[YX1]  132:            mov     AL,1Øh
[ZR1]  133:            shl     AL,CL
[7Z2]  134:            test    motwert,AL
[ZXØ]  135:            pushf
[KN2]  136:            or      AL,motwert
[FY2]  137:            mov     motwert,AL
[6L1]  138:            or      AL,DL
[SS1]  139:            or      AL,ØCh
[R72]  14Ø:            mov     DX,Ø3F2h
[6V1]  141:            out     DX,AL
[LSØ]  142:            popf
[A92]  143:            jz      waitmon
[6PØ]  144:            sti
[3PØ]  145:            ret
[294]  146: waitmon:   mov     AL,montime
[5X2]  147:            mov     counter,AL
[6PØ]  148:            sti
[4S1]  149:            jmp     wait
[WG2]  15Ø: motoron    endp
[ØØØ]  151:
[5K3]  152: motoroff   proc    near
[9M4]  153:   ;Laufwerkmotor ausschalten
[GA3]  154:   ;DL=Laufwerknummer
[BH1]  155:            and     DL,3
[NT1]  156:            mov     CL,DL
[YX1]  157:            mov     AL,1Øh
[ZR1]  158:            shl     AL,CL
[7Z2]  159:            test    motwert,AL
[HB2]  16Ø:            jz      exitmoff
[ZC1]  161:            not     AL
[CV2]  162:            and     motwert,AL
[XV2]  163:            mov     AL,motwert
[SS1]  164:            or      AL,ØCh
[XF1]  165:            push    AX
[BØ3]  166:            mov     AL,mofftime
[5X2]  167:            mov     counter,AL
[ER1]  168:            call    wait
[RD1]  169:            pop     AX
```

»fdc.asm« zeigt Ihnen die Programmierung des Diskettencontrollers

```
[EMØ]  17Ø:              cli
[R72]  171:              mov     DX,Ø3F2h
[6V1]  172:              out     DX,AL
[6PØ]  173:              sti
[V82]  174: exitmoff:    ret
[2H2]  175: motoroff     endp
[ØØØ]  176:
[LG3]  177: drvrekal     proc    near
[D44]  178:   ;Laufwerk rekalibrieren
[GA3]  179:   ;DL=Laufwerknummer
[8D1]  18Ø:   ;DH=Kopf
[8K3]  181:   ;=> Carry, wenn Fehler
[AG2]  182:              call    seldrive
[HA3]  183:              mov     CX,rekalretry
[P23]  184: rekloop:     push    CX
[CM1]  185:              mov     AL,7
[UG2]  186:              call    sendbyte
[592]  187:              mov     AL,drhd
[ZM2]  188:              call    startexec
[Ø63]  189:              mov     AL,headsetdel
[5X2]  19Ø:              mov     counter,AL
[ER1]  191:              call    wait
[DM1]  192:              mov     AL,8
[UG2]  193:              call    sendbyte
[FP2]  194:              call    getresult
[VD1]  195:              pop     CX
[VF2]  196:              jz      dontrekal
[7G2]  197:              loop    rekloop
[ENØ]  198:              stc
[3PØ]  199:              ret
[BA2]  2ØØ: dontrekal:   clc
[3PØ]  2Ø1:              ret
[GF2]  2Ø2: drvrekal     endp
[ØØØ]  2Ø3:
[423]  2Ø4: seek         proc    near
[FC2]  2Ø5:   ;Spur anfahren
[8D1]  2Ø6:   ;DH=Kopf
[M52]  2Ø7:   ;DL=Laufwerk
[RG1]  2Ø8:   ;CH=Track
[8K3]  2Ø9:   ;=> Carry, wenn Fehler
[1G1]  21Ø:              push    CX
[AG2]  211:              call    seldrive
[9Z1]  212:              mov     AL,ØFh
[UG2]  213:              call    sendbyte
[592]  214:              mov     AL,drhd
[UG2]  215:              call    sendbyte
[RD1]  216:              pop     AX
[3S1]  217:              xchg    AH,AL
[ZM2]  218:              call    startexec
[Ø63]  219:              mov     AL,headsetdel
[5X2]  22Ø:              mov     counter,AL
[ER1]  221:              call    wait
[DM1]  222:              mov     AL,8
[UG2]  223:              call    sendbyte
[5R2]  224:              jmp     getresult
[112]  225: seek         endp
[ØØØ]  226:
[B53]  227: newintØ8     proc    far
[XF1]  228:              push    AX
[KR4]  229:              cmp     byte ptr CS:counter,Ø
[SF2]  23Ø:              jz      exitintØ8
[WG4]  231:              dec     byte ptr CS:counter
[6K3]  232: exitintØ8:   mov     AL,6Øh
[LZ1]  233:              out     2Øh,AL
[RD1]  234:              pop     AX
[GRØ]  235:              iret
[DD2]  236: newintØ8     endp
[ØØØ]  237:
[S53]  238: newintØe     proc    far
[XF1]  239:              push    AX
[5N3]  24Ø:              mov     CS:fdcstatus,-1
[FY1]  241:              mov     AL,66h
[LZ1]  242:              out     2Øh,AL
[RD1]  243:              pop     AX
[GRØ]  244:              iret
[UD2]  245: newintØe     endp
[ØØØ]  246:
[NH3]  247: seldrive     proc    near
[2N3]  248:   ;Laufwerk selektieren
[8D1]  249:   ;DH=Kopf
[M52]  25Ø:   ;DL=Laufwerk
[614]  251:   ;=> Carry, wenn Motor des
[WH3]  252:   ;   Laufwerks nicht an
[ØØØ]  253:
[BH1]  254:              and     DL,3
[NT1]  255:              mov     CL,DL
[YX1]  256:              mov     AL,1Øh
[ZR1]  257:              shl     AL,CL
[7Z2]  258:              test    motwert,AL
[2Ø2]  259:              jnz     motok
[ENØ]  26Ø:              stc
[3PØ]  261:              ret
[G43]  262: motok:       mov     AL,DL
[KN2]  263:              or      AL,motwert
[SS1]  264:              or      AL,ØCh
[EMØ]  265:              cli
[3G1]  266:              push    DX
[R72]  267:              mov     DX,Ø3F2h
[6V1]  268:              out     DX,AL
[XD1]  269:              pop     DX
[6PØ]  27Ø:              sti
```

```
[9T1]  271:              mov     AL,DH
[3H1]  272:              and     AL,1
[NK1]  273:              shl     AL,1
[NK1]  274:              shl     AL,1
[6L1]  275:              or      AL,DL
[CB2]  276:              mov     drhd,AL
[PLØ]  277:              clc
[3PØ]  278:              ret
[KG2]  279: seldrive     endp
[ØØØ]  28Ø:
[T23]  281: wait         proc    near
[755]  282:   ;wartet bestimmte Zeiteinheit
[2V2]  283:   ;in 1/1ØØ Sekunden
[ØØØ]  284:
[754]  285: waitloop:    cmp     counter,Ø
[ØK2]  286:              jnz     waitloop
[3PØ]  287:              ret
[P12]  288: wait         endp
[ØØØ]  289:
[BE3]  29Ø: testfdc      proc    near
[VR3]  291:   ;Testen, ob FDC bereit
[3Z1]  292:   ;DX=Ø3F4hex
[PV1]  293:              xor     CX,CX
[T33]  294: tfloop:      in      AL,DX
[NK1]  295:              shl     AL,1
[6M1]  296:              jc      fdcok
[S62]  297:              loop    tfloop
[PY1]  298:              jmp     error
[FW1]  299: fdcok:       ret
[8D2]  3ØØ: testfdc      endp
[ØØØ]  3Ø1:
[5K3]  3Ø2: sendbyte     proc    near
[BV4]  3Ø3:   ;Kommandobyte an FDC schicken
[TW2]  3Ø4:   ;AL=Kommandobyte
[2T1]  3Ø5:              mov     AH,AL
[Z72]  3Ø6:              mov     DX,Ø3F4h
[ØC2]  3Ø7:              call    testfdc
[NK1]  3Ø8:              shl     AL,1
[KM2]  3Ø9:              jnc     startbyte
[PY1]  31Ø:              jmp     error
[CH3]  311: startbyte:   mov     AL,AH
[SB1]  312:              inc     DX
[6V1]  313:              out     DX,AL
[813]  314:              mov     CX,fdcdelay
[EA3]  315: delay:       loop    delay
[3PØ]  316:              ret
[2H2]  317: sendbyte     endp
[ØØØ]  318:
[AM3]  319: startexec    proc    near
[HN4]  32Ø:   ;Ausführungsphase starten
[DC4]  321:   ;AL=letztes Kommandobyte
[GZ2]  322:              mov     fdcstatus,Ø
[UG2]  323:              call    sendbyte
[PV1]  324:              xor     CX,CX
[853]  325:              mov     AL,exeretry
[LH4]  326: waitfdc:     cmp     fdcstatus,-1
[AZ1]  327:              jz      execok
[YC2]  328:              loop    waitfdc
[991]  329:              dec     AL
[LC2]  33Ø:              jnz     waitfdc
[PY1]  331:              jmp     error
[832]  332: execok:      ret
[7L2]  333: startexec    endp
[ØØØ]  334:
[UN3]  335: getresult    proc    near
[K64]  336:   ;Ergebnisbytes auslesen
[G82]  337:   ;keine Eingabe
[VF5]  338:   ;=>  [result]=Anz. Ergebnisbytes
[UA4]  339:   ;    [result+1]=1. Ergebnis
[UU4]  34Ø:   ;bis [result+x]=x. Ergebnis
[MØ5]  341:   ;Zero, wenn Operation erfolgr.
[ULØ]  342:              cld
[EU3]  343:              mov     DI,offset result
[Z72]  344:              mov     DX,Ø3F4h
[ØT1]  345:              xor     AH,AH
[7N1]  346:              mov     SI,7
[PE1]  347:              push    DI
[BA1]  348:              inc     DI
[PV1]  349:              xor     CX,CX
[U73]  35Ø: resloop:     in      AL,DX
[NK1]  351:              shl     AL,1
[CB2]  352:              jc      resultok
[7H2]  353:              loop    resloop
[MS3]  354: errresult:   jmp     error
[B93]  355: resultok:    shl     AL,1
[BM2]  356:              jnc     endresult
[AM1]  357:              or      SI,SI
[7L2]  358:              jz      errresult
[SB1]  359:              inc     DX
[6L1]  36Ø:              in      AL,DX
[NXØ]  361:              stosb
[LA1]  362:              dec     DX
[4A1]  363:              dec     SI
[2A1]  364:              inc     AH
[CB3]  365:              mov     CX,resuldelay
[XF3]  366: delay1:      loop    delay1
[HB3]  367:              jmp     short resloop
[323]  368: endresult:   pop     DI
[XB2]  369:              mov     [DI],AH
[HD4]  37Ø:              test    byte ptr [DI+1],ØCØh
[3PØ]  371:              ret
```

```
[RM2]  372: getresult  endp
[ØØØ]  373:
[WE3]  374: progdma    proc    near
[W75]  375:   ;DMA-Kontroller programmieren
[U14]  376:   ;[dmacom]=DMA-Kommando:
[CR2]  377:   ;          46hex=Lesen
[U24]  378:   ;          42hex=Verifizieren
[NC3]  379:   ;          4Dhex=Schreiben
[MP2]  38Ø:   ;AH= Pageadresse
[463]  381:   ;BX= Offsetadresse
[5Z2]  382:   ;CX= Anzahl Bytes
[BF2]  383:             mov     AL,dmacom
[GV1]  384:             mov     DX,AX
[BM1]  385:             mov     AL,6
[212]  386:             out     Øah,AL
[EMØ]  387:             cli
[812]  388:             out     ØCh,AL
[FT1]  389:             mov     AL,BL
[9M1]  39Ø:             out     4,AL
[SC2]  391:             jmp     short n1
[9C3]  392: n1:         jmp     short n2
[GC3]  393: n2:         jmp     short n3
[RR2]  394: n3:         mov     AL,BH
[9M1]  395:             out     4,AL
[9T1]  396:             mov     AL,DH
[2Ø2]  397:             out     81h,AL
[HA1]  398:             dec     CX
[GT1]  399:             mov     AL,CL
[BM1]  4ØØ:             out     5,AL
[VC2]  4Ø1:             jmp     short n4
[YC3]  4Ø2: n4:         jmp     short n5
[5D3]  4Ø3: n5:         jmp     short n6
[4S2]  4Ø4: n6:         mov     AL,CH
[BM1]  4Ø5:             out     5,AL
[6PØ]  4Ø6:             sti
[HT1]  4Ø7:             mov     AL,DL
[512]  4Ø8:             out     ØBh,AL
[7M1]  4Ø9:             mov     AL,2
[212]  41Ø:             out     ØAh,AL
[3PØ]  411:             ret
[TD2]  412: progdma    endp
[ØØØ]  413:
[HT3]  414: optsegment proc    near
[E15]  415:   ;Pufferadresse optimieren und
[F94]  416:   ;auf DMA-Überlauf testen
[A45]  417:   ;ES=Segmentadresse des Buffers
[4P4]  418:   ;DI=Offsetadresse des Buffer
[WT2]  419:   ;CX=Anzahl Bytes
[U43]  42Ø:   ;=> AH = Pageadresse
[AD3]  421:   ;    BX = Offsetadresse
[7H4]  422:   ;    Carry, wenn DMA-Überlauf
[1G1]  423:             push    CX
[4V1]  424:             mov     AX,ES
[CV1]  425:             mov     BX,AX
[K22]  426:             and     AH,ØFØh
[DM1]  427:             mov     CL,4
[BS1]  428:             shr     AH,CL
[5T1]  429:             shl     BX,CL
[GP1]  43Ø:             add     BX,DI
[MF1]  431:             adc     AH,Ø
[4U1]  432:             mov     DI,BX
[VD1]  433:             pop     CX
[CN2]  434:             test    dmacom,2Øh
[2T1]  435:             jz      dirup
[LS1]  436:             sub     DI,CX
[3PØ]  437:             ret
[113]  438: dirup:      add     DI,CX
[3PØ]  439:             ret
[ES2]  44Ø: optsegment endp
[ØØØ]  441:
[B73]  442: error       proc    near
[YY4]  443:   ;Diese Routine wird immer beim
[WU4]  444:   ;Auftreten eines Fehlers auf-
[BF1]  445:   ;gerufen.
[EMØ]  446:             cli
[F52]  447:             mov     SP,Ø1ØØh
[6PØ]  448:             sti
[AW1]  449:             dec     retry
[VM1]  45Ø:             jz      quit
[TV1]  451:             xor     DX,DX
[8F2]  452:             call    drvrekal
[343]  453:             jmp     short repeat
[NØ3]  454: quit:       xor     DL,DL
[UG2]  455:             call    motoroff
[GU2]  456:             call    fdcrestore
[8U2]  457: err:        mov     AH,9
[GB3]  458:             mov     DX,offset msg2
[3K1]  459:             int     21h
[B62]  46Ø:             mov     AX,4CØØh
[3K1]  461:             int     21h
[862]  462: error       endp
[ØØØ]  463:
[LZ2]  464: ; Initialisierung
[433]  465: init        proc    near
[EMØ]  466:             cli
[2V1]  467:             mov     AX,CS
[1V1]  468:             mov     DS,AX
[3V1]  469:             mov     ES,AX
[ZV1]  47Ø:             mov     SS,AX
[F52]  471:             mov     SP,Ø1ØØh
[6PØ]  472:             sti
[YA2]  473:             mov     retry,5
```

```
[445]  474:   ;fünf Versuche beim Lesen des
[YF1]  475:   ;Sektors
[MK2]  476:             mov     intflag,Ø
[9P2]  477:             mov     motwert,Ø
[125]  478:   ;beide Werte immer auf Ø setzen
[MT1]  479:             xor     AL,AL
[EN1]  48Ø:             mov     CX,1
[TV1]  481:             xor     DX,DX
[2R3]  482:             mov     BX,offset buffer
[FK1]  483:             int     25h
[185]  484:   ;Bootsektor durch MS-DOS lesen
[FD2]  485:             jnc     perform
[LSØ]  486:             popf
[3R1]  487:             jmp     err
[NG2]  488: perform:   popf
[PF1]  489:             push    ES
[5Z1]  49Ø:             mov     AX,4Øh
[3V1]  491:             mov     ES,AX
[HE4]  492: waitmoff:   mov     AL,ES:[3Fh]
[6W1]  493:             and     AL,ØFh
[MG2]  494:             jnz     waitmoff
[255]  495:   ;Warten, bis Laufwerkmotor aus
[PV1]  496:             xor     CX,CX
[UD3]  497: waitx:      loop    waitx
[GD1]  498:             pop     ES
[TB2]  499:             call    fdcinit
[ZS4]  5ØØ:   ;Interrupts initialisieren
[XT1]  5Ø1:             xor     DL,DL
[MF2]  5Ø2:             call    motoron
[8Y4]  5Ø3:   ;Motor Laufwerk A einschalten
[N73]  5Ø4: repeat:     xor     CH,CH
[TV1]  5Ø5:             xor     DX,DX
[TP1]  5Ø6:             call    seek
[PS2]  5Ø7:             ;Track Ø anfahren
[X12]  5Ø8:             jnc     seekok
[PY1]  5Ø9:             jmp     error
[SY3]  51Ø: seekok:     mov     dmacom,46h
[4D5]  511:   ;DMA-Kommando ¤Lesen³ setzen
[594]  512:             mov     DI,offset buffer+2ØØh
[AX1]  513:             mov     CX,512
[6X2]  514:             call    optsegment
[5T3]  515:   ;Bufferadresse für DMA
[442]  516:   ;optimieren
[WE2]  517:             jnc     startcom
[PY1]  518:             jmp     error
[XZ3]  519: startcom:   call    progdma
[W75]  52Ø:   ;DMA-Kontroller programmieren
[H62]  521:             mov     AL,ØE6h
[UG2]  522:             call    sendbyte
[592]  523:             mov     AL,drhd
[UG2]  524:             call    sendbyte
[MT1]  525:             xor     AL,AL
[UG2]  526:             call    sendbyte
[UG2]  527:             call    sendbyte
[EA1]  528:             inc     AL
[UG2]  529:             call    sendbyte
[7M1]  53Ø:             mov     AL,2
[UG2]  531:             call    sendbyte
[6M1]  532:             mov     AL,1
[UG2]  533:             call    sendbyte
[1Z1]  534:             mov     AL,2Ah
[UG2]  535:             call    sendbyte
[ØC5]  536:   ;ersten acht Kommandobytes für
[3Y5]  537:   ;¤Sektor lesen³ an FDC schicken
[382]  538:             mov     AL,ØFFh
[ZM2]  539:             call    startexec
[795]  54Ø:   ;letztes Kommandobyte schicken
[CB5]  541:   ;und Ausführungsphase starten
[FP2]  542:             call    getresult
[R44]  543:   ;Ergebnisphase auslesen
[181]  544:             jz      ok
[PY1]  545:             jmp     error
[PV2]  546: ok          label   near
[A35]  547:   ;bei Schreibzugriffen folgende
[4A4]  548:   ;Sequenz implementieren:
[T24]  549: ;           mov     CX,writedelay
[FL3]  55Ø: ;wloop:     loop    wloop
[XT1]  551:             xor     DL,DL
[UG2]  552:             call    motoroff
[EY4]  553:   ;Motor Laufwerk A ausschalten
[GU2]  554:             call    fdcrestore
[9G4]  555:   ;Interrupts restaurieren
[DB3]  556:             mov     DX,offset msg1
[2M1]  557:             mov     AH,9
[3K1]  558:             int     21h
[B62]  559:             mov     AX,4CØØh
[3K1]  56Ø:             int     21h
[ØØØ]  561:
[RN3]  562: ;Datenbereich für Demo
[M65]  563: msg1        db Ødh,Øah,"Zugriff Ok.",Ødh
[S61]  564:             db Øah,"$"
[RØ6]  565: msg2        db Ødh,Øah,"Fehler aufgetreten."
[SS1]  566:             db Ødh,Øah,"$"
[ØØØ]  567:
[522]  568: buffer      db ?
[122]  569: init        endp
[ØØØ]  57Ø:
[112]  571: CODE        ends
[4W1]  572:             end     start
```

»fdc.asm« zeigt Ihnen die Programmierung des Diskettencontrollers

Treiber-Programmierung

Daten im Erweiterungsspeicher

Daß die 640 KByte, die MS-DOS unterstützt, den Anforderungen heutiger Programme nicht mehr gerecht werden, ist ein alter Hut. Für große Datenmengen bleibt nur, diese in einen externen Speicher oder den Erweiterungsspeicher auszulagern. Ein entsprechender Treiber bietet hier eine interessante Alternative, den Arbeitsspeicher für die Datenaufnahme um die Größe des Erweiterungsspeichers zu erweitern.

Kaufen Sie sich einen Rechner mit 2 oder 4 MByte Arbeitsspeicher, so ist das vorerst ein trügerischer »Speicherreichtum«. MS-DOS beachtet die vielen tausend KByte nämlich einfach nicht. Um Daten in den Erweiterungsspeicher auszulagern, benötigen Sie einen Treiber, der Schreib- und Leseoperationen in den beziehungsweise aus dem Erweiterungsspeicher unterstützt. Des weiteren müssen die verwendeten Programme diese Funktionen auch nutzen.

Der im folgenden beschriebene Treiber sendet und empfängt Daten von 2 Byte bis zu einer Satzlänge von 65534 (FFFEhex) Byte vom Arbeitsspeicher zum Erweiterungsspeicher und umgekehrt. Er bietet also die Fähigkeit, ganze Segmente auszulagern.

Der »EXTended-Memory-TReiBer« (»EXTM_TRB«) sendet und empfängt die Daten über die MS-DOS-Funktion »ioctl«, der Treiber selbst verwendet den BIOS-Interrupt 15hex. Der Treiber hat als »extm_trb.sys« eine Größe von zirka 1700 Byte und belegt nach seiner Installation etwas mehr als 1 KByte. Das bedeutet eine geringfügige Verringerung des Arbeitsspeichers, mit der Sie je nach Erweiterungsspeicher bis zu 16 MByte hinzugewinnen.

Als erstes nehmen Sie den Treiber in die »config.sys« auf, von der aus er sich beim Systemstart in die MS-DOS-Treiberreihe einbindet. Da er bei der Installation überprüft, ob sich eine RAM-Disk im Erweiterungsspeicher befindet, muß diese vor dem Treiber in der »config.sys« stehen. Der Treiber erkennt bis jetzt allerdings nur die RAM-Disk von MS-DOS 3.3 und auch keine anderen Treiber, die auf den Erweiterungsspeicher zugreifen und sich dort Platz reservieren.

Der Treiber installiert sich nur, wenn er festgestellt hat, ob es sich um einen AT handelt. Sodann stellt er die Größe des Erweiterungsspeichers fest, holt die ersten 256 Byte des Erweiterungsspeichers und prüft, ob eine RAM-Disk installiert ist. Ist dies der Fall, entnimmt er die Speicherangaben über die RAM-Disk dem EMM-Control-Block (EMM = Extended Memory Management), rechnet diese, wenn notwendig, in Bytes um und übernimmt sie in den Treiber. Ist keine RAM-Disk vorhanden, schreibt der Treiber seinen eigenen Control-Block in den Erweiterungsspeicher und merkt sich die Größe des freien Erweiterungsspeichers.

Als Anwendungsbeispiel betrachten Sie bitte das Programm »extm_tst.c« (Listing 1) zum Übertragen von Daten in den Erweiterungsspeicher. Als erstes stellt das Programm den Arbeitspuffer »buf« bereit, aus dem der Treiber die Daten empfängt oder dorthin sendet. Weiter enthält es folgende Struktur:

```
struct{extm_adr[5]
       unsigned int len
       int key
}
```

»extm_adr[5]« nimmt die Adresse des Erweiterungsspeichers auf, »len« die Länge des zu übertragenden Datensatzes, und der Schlüssel »key« kann ein Identifizierungsmerkmal enthalten, zum Beispiel eine Kunden- oder Artikelnummer, über die die zum Datensatz gehörende Adresse des Erweiterungsspeichers ermittelt wird.

Die Verbindung zum Treiber stellt die Funktion »_open« her, die ein Handle liefert. Dann setzt die Ioctl-Funktion 1 für den Treiber einen Zeiger auf »buf«. Aufgrund der Längenangabe 1 erkennt der Treiber, daß es sich um eine Initialisierung handelt und trägt den übergebenen Zeiger in den Puffer »(ram_buf)« ein.

Diese Ioctl-Funktion ist immer erforderlich, da dem Treiber sonst die Adresse des Puffers »buf« nicht bekannt ist.

Als nächstes werden die Daten im Puffer bereitgestellt. Nun überträgt die Ioctl-Funktion 3 die Daten zum Treiber, wozu Sie dieser als weiteren Parameter das Handle und einen Zeiger auf »extm_adr« (»tabelle[i].extm_adr«) mitgeben. Der Treiber trägt nach seiner Arbeit in »extm_adr« das Kennzeichen »L« für ein späteres Lesen der Daten und die 24 Bit große Adresse, bei der er die Daten gespeichert hat, ein. Diese Adresse steht in drei Bytes (char), die nicht manipuliert werden dürfen, da sie beim Lesen in genau der richtigen Reihenfolge wieder entnommen werden. Im vierten Byte wird 0 eingetragen. In »tabelle[i].len« wird die Länge des Datensatzes übergeben.

Zum Empfangen der Daten aus dem Erweiterungsspeicher überträgt die Ioctl-Funktion die Daten wieder vom Treiber in den Puffer »buf«. Mit den Parametern »handle«, »2« für Empfangen, dem Zeiger auf die Adresse (»tabelle[i].extm_adr«) sowie der Längenangabe liest »ioctl« die Daten aus dem Erweiterungsspeicher. Sie können über das Schlüsselfeld »key« den richtigen Datensatz ermitteln oder alle Sätze der Reihe nach auslesen. Der Treiber erkennt am Kennzeichen »L«, daß er diesen Datensatz auslesen soll und holt sich dessen Adresse. Möchten Sie einen Satz nicht lesen, so können Sie das Kennzeichen »L« vorübergehend oder ganz durch ein anderes Zeichen ersetzen. Dieser Satz wird dann nicht erkannt.

Sie können vorher eingetragene Daten auch nachträglich ändern, wobei sich die Satzlänge nicht ändern sollte, sich aber auf gar keinen Fall vergrößern darf. Der Datensatz wird wie beschrieben ermittelt, geändert und im Puffer »buf« wieder bereitgestellt.

Die Funktion »ioctl« ist die Schnittstelle zum Treiber

Ein Zurückschreiben desselben Datensatzes führen Sie mit der Unterfunktion 3 der Ioctl-Funktion durch. Aufgrund des Kennzeichens »L« erkennt der Treiber, daß bereits eine Adresse bekannt ist und schreibt den Datensatz an dieselbe Stelle in den Erweiterungsspeicher zurück. Zum Schluß eines Programms rufen Sie zum letztenmal die Unterfunktion 3 mit dem Kennzeichen »A« und der Länge 1 auf. Mit dem Kennzeichen »A« (Anfang) stellt der Treiber die Anfangsadressen des Erweiterungsspeichers wieder her.

Der Treiber überprüft jedesmal vor dem Schreiben, ob noch genügend Platz im Erweiterungsspeicher vorhanden ist. Wenn nein, ertönt ein Piepton, und er schreibt die Meldung

»@ EXTMEMORY VOLL«

in den vom Programm übergebenen Puffer. Auch hier können Sie im Anwenderprogramm das erste Zeichen »@« abfragen und entsprechend reagieren.

Wie ein Anwenderprogramm die Tabellen oder Strukturen erzeugt oder verwaltet, liegt ganz im Ermessen des Anwenders. Es ist nur wichtig, daß beim Lesen und Schreiben die Adressen und Längenangaben stimmen. So übergibt zum Beispiel der Aufruf von

```
ioctl(handle,3,buffer,1);
```

vor jeder Leseoperation eine andere Pufferadresse (»buffer«), in die der Treiber dann den Datensatz schreibt.

Der Treiber berücksichtigt eine vorhandene RAM-Disk

Der Treiber »extm_trb.sys« ist als Zeichentreiber konzipiert und hat den Namen »EXTMMTRB«. Beim Starten wird von MS-DOS als erstes die Initialisierungsroutine »init« des Treibers aufgerufen.

- Die Prozedur »init« überprüft als erstes, ob der Rechner ein AT ist. Nur dann installiert sich der Treiber, was er mit einer Installationsmeldung bekundet. Anschließend speichert er die Adresse des Puffers »buf« in »ram_buf« und ruft die Prozedur »ext_mem« auf. Diese Prozedur schreibt oder liest Daten aus dem Erweiterungsspeicher, hier die ersten 256 Byte ab der Adresse 100000hex. Das ist die Startadresse des Erweiterungsspeichers, bei der, für den Fall, daß eine RAM-Disk installiert ist, der EMM-Control-Block von MS-DOS 3.3 gespeichert ist.
- »extm_init« ist die Initialisierungsroutine des Treibers, die von «init« aufgerufen wird. »extm_init« ermittelt als erstes die Speichergröße des Erweiterungsspeichers, rechnet diese in Bytes um und legt sie in »extm_total« ab. Dann vergleicht die Routine die ersten 31 Byte der Variablen »buf« mit »buf_vergl«, um festzustellen, ob eine RAM-Disk installiert ist. Ist keine RAM-Disk vorhanden, speichert »extm_init« die Adresse des eigenen Control-Blocks in »ram_buf« und schreibt diesen Block in den Erweiterungsspeicher.

Die Variable »richtung« teilt der Prozedur »ext_mem« mit, ob sie lesen oder schreiben soll. Die Treibervariable »start_adr« ist für die Initialisierung mit Adresse 100000hex vorbelegt. »extm_init« addiert die Satzlänge auf die Startadresse »start_adr«, zieht diesen Wert von »extm_total« ab und speichert das Ergebnis danach in »extm_frei«. Das Ergebnis kommt zudem nach »z_extm_frei«, um beim Zurücksetzen (Kennzeichen »A«) die Anfangswerte wieder in »extm_frei« einzutragen.

Ist eine RAM-Disk vorhanden, werden die Werte aus dem EMM-Control-Block von MS-DOS ausgelesen und entsprechend für den Treiber berechnet. Im Puffer »buf« befindet sich der EMM-Control-Block, bei »buf+70« (Startadresse des Puffers plus 70) und »buf+68« die Startadresse der RAM-Disk. »extm_init« schreibt die Startadresse der RAM-Disk in »AL:BX« und trägt sie in »start_adr« ein.

Als nächstes liest die Routine die Größe der RAM-Disk in KByte aus dem Speicherwort bei »buf+72«, rechnet diese in Bytes um und addiert diesen Wert zu »start_adr«.

Bei »buf+52« befindet sich der Wert des noch freien Erweiterungsspeichers, ebenfalls in KByte. Nach der Umrechnung in Bytes landet das Ergebnis in »extm_frei« und in »z_extm_frei« (für das Rücksetzen auf den Anfang). Als letztes wird die Startadresse »start_adr« in die Variable »anf_adr« eingetragen. Auch diese Anfangsadresse wird beim Zurücksetzen in die Ausgangssituation gebracht. Nach dem Restaurieren der anfangs gesicherten Register ist die Initialisierung abgeschlossen.

- »inter« lädt die Adresse des Parameterdatenblocks nach »ES:DI« und ruft die Funktionen des Treibers auf. Anschließend schreibt sie das Statusbyte und übergibt die Kontrolle wieder an MS-DOS.

Eine Interruptroutine ruft die Treiberfunktion auf

- Die Prozedur »schreiben« dient zum Schreiben vom Arbeitsspeicher in den Erweiterungsspeicher. Als erstes speichert sie die Anzahl der Zeichen aus dem übergebenen Datenblock im Register »CX« und testet diesen Wert auf kleiner 1. Ist das der Fall, handelt es sich um eine Initialisierung. Die Routine verzweigt zur Marke »buffer« und prüft, ob das Kennzeichen »A« übergeben wurde: Wenn nicht, liest »schreiben« die Adresse des Arbeitspuffers aus dem Datenblock und trägt diese in die Variable »ram_buf« ein. Bei Kennzeichen »A« wird die Prozedur »kennz_a« aufgerufen, die die Anfangswerte aus »z_start_adr« und »z_extm_frei« wieder in die Variablen »start_adr« beziehungsweise »extm_frei« ein-

Toolboxen für Programmierer gesucht

Haben Sie eine Toolbox in C, Pascal, Basic, dBase, Clipper oder Assembler geschrieben? Beispielsweise schnelle, universelle Grafikroutinen, eine Textverarbeitungs- oder eine Datenbank-Toolbox? Dann sollten Sie sich schnellstmöglich mit uns in Verbindung setzen. Schicken Sie uns unverbindlich eine Diskette mit Ihrem Programm und einen Ausdruck der Dokumentation.

Falls Sie sich vorweg informieren möchten: Zwischen 15.00 und 17.00 Uhr stehen wir Ihnen an allen Werktagen unter der Rufnummer (0 81 21) 7 69-0 zur Verfügung.

Schicken Sie Ihre Unterlagen an folgende Adresse:

Redaktion DOS International, »DOS Software«,
Gruberstraße 46c, 8011 Poing

trägt. Der Treiber befindet sich damit wieder in der Ausgangsposition.
Bei einer Satzlänge größer 1 wird geprüft, ob die Satzlänge ungerade ist. Sie wird dann eventuell um 1 erhöht (so daß ein gerader Wert vorliegt) und in der Variablen »satz_len« abgelegt. Als nächstes wird die Prozedur »frei« aufgerufen, die prüft, ob der Datensatz noch im Erweiterungsspeicher Platz hat. Wenn ja, wird die Satzlänge von »extm_frei« abgezogen und der neue Wert in »extm_frei« gespeichert. Ansonsten wird die Satzlänge auf 0 gesetzt, die Meldung

»@ EXTMEMORY VOLL«

in den Arbeitspuffer des aufrufenden Programms geschrieben und ein Piepton erzeugt. Bei der Rückkehr aus »frei«, prüft der Treiber, ob die Satzlänge 0 ist, und verläßt bei Gleichheit die Prozedur »schreiben« mit einer Fehlerkennung im Statusbyte. Andernfalls geht es bei »sp_ok« mit einem Aufruf der Prozedur »teilen« weiter, die die Satzlänge halbiert und in »satz_len_w« speichert (für die Übertragung in Worten in den Erweiterungsspeicher).

Der Treiber gibt Fehlermeldungen zurück

»schreiben« ruft nun die Prozedur »ext_mem« auf, die aus dem Inhalt der Variablen »richtung« (3) erkennt, daß sie die Daten in den Erweiterungsspeicher schreiben soll. Bei der Rückkehr von »ext_mem« prüft »schreiben« nochmals auf das Kennzeichen »L«, um, wenn nötig, über die Prozedur »adr_ber« die aktuelle Startadresse wieder in »start_adr« einzutragen.

- Das Gegenstück zu »schreiben« bildet »lesen«. Diese Prozedur überprüft als erstes das Kennzeichen auf »L« und springt bei Ungleichheit direkt zum Ende, ohne den Datensatz zu lesen. Bei Gleichheit lädt sie die Anzahl der Zeichen in das CX-Register. Ist die Anzahl der zu übertragenden Zeichen gleich 0, endet die Prozedur. Sonst »begradigt« sie, wenn nötig, die Anzahl der Zeichen. Dann wird die Satzlänge in der Variablen »satz_len« gespeichert und die Prozedur »teilen« aufgerufen, die die Satzlänge für den BIOS-Interrupt 15hex, Funktion 87hex, halbiert, da dieser nur wortweise liest. Die Prozedur »les_adr« speichert die aktuelle Startadresse in der Variablen »z_start_adr« und holt die Adresse des zu lesenden Datensatzes aus dem übergebenen Datenblock. Diese Adresse wird nun für die Prozedur »ext_mem« in die Variable »start_adr« eingetragen. Nach der Rückkehr aus »les_adr« wird die Richtung (hier »2«) in die Variable »richtung« eingetragen und die Prozedur »ext_mem« aufgerufen. »ext_mem« erkennt an der Variablen »richtung«, daß es sich um einen Lesevorgang handelt und holt den Datensatz aus dem Erweiterungsspeicher. Die Prozedur »adr_ber« stellt die aktuelle Startadresse wieder her.
- »strat« speichert die Adresse des von MS-DOS in »ES:BX« übergebenen Parameterdatenblocks in »db_ptr«.
- »dummy« löscht das Beschäftigt-Bit.
- »dir_no« fängt falsch aufgerufene Funktionen ab. Die Routine setzt das Fehlerbit im Statusbyte und gibt einen Fehlercode zurück.
- »besch« setzt das Beschäftigt-Bit.
- »update« bringt die »start_adr« nach »z_start_adr« und trägt die Adresse des Erweiterungsspeichers aus dem Programm in »start_adr« ein.
- »adr_ber« trägt die Adresse aus »z_start_adr« wieder in »start_adr« ein und berichtigt diese damit.
- »frei« kontrolliert in der übergebenen Adresse des Anwenderprogramms das erste Zeichen auf das Kennzeichen »L« und ruft bei Gleichheit die Prozedur »update« auf. Ist das erste Byte ungleich «L«, ist es ein neuer Datensatz, und die Routine »frei« überprüft, ob noch genügend Platz im Erweiterungsspeicher vorhanden ist. Ist dies der Fall, so wird die Satzlänge von »extm_frei« subtrahiert.
- »teilen« halbiert die Satzlänge und trägt sie in »satz_len_w« ein. Die Prozedur »ext_mem« überträgt dann den Datensatz wortweise.
- »st_adr« schreibt die aktuelle Startadresse des Erweiterungsspeichers mit dem Kennzeichen »L« in die vom Anwender-Programm übergebenen Variablen und addiert die Satzlänge zu »start_adr«. Somit steht in »start_adr« die neue Adresse des Erweiterungsspeichers, ab der ein neuer Datensatz gespeichert werden kann.
- »les_adr« bringt die aktuelle Startadresse aus »start_adr« nach »z_start_adr«, liest dann die Adresse aus der Variablen im Anwenderprogramm aus und trägt diese in »start_adr« ein.
- »kennz_a« stellt die Anfangsadressen wieder her, damit beim Verlassen des Anwenderprogramms die initialisierten Adressen wieder bereit stehen.
- »ext_mem« errechnet die 24-Bit-Start- und Zieladressen der GDT. Je nach dem Inhalt der Variablen »richtung« werden diese in »start_hi« und »start_lo« beziehungsweise »ziel_hi« und »ziel_lo« eingetragen.

Das Programm »extm_trb.asm« (Listing 2) enthält alle beschriebenen Funktionen und stellt somit einen kompletten Treiber für den Erweiterungsspeicher dar. Durch den Aufruf von

```
tasm extm_trb
```

erhalten Sie eine Obj-Datei, aus der Sie mit

```
tlink extm_trb
```

eine Exe-Datei erzeugen. Diese wandeln Sie anschließend mit

```
exe2bin extm_trb.exe
extm_trb.sys
```

in die benötigte Treiberdatei um. »extm_trb.asm bindet die beiden Dateien »pushall.mac« (Listing 3) sowie »popall.mac« (Listing 4) ein, die Routinen zum Sichern und Restaurieren der Register enthalten.
In Zukunft können Sie die Daten Ihrer Programme in den Erweiterungsspeicher auslagern, so daß mehr Platz für Ihre Programme bleibt. Außer C können Sie jede andere Sprache verwenden, sie muß lediglich in der Lage sein, die Funktion »ioctl« aufzurufen.
Um Mißverständnissen vorzubeugen, sei erwähnt, daß dieser Treiber lediglich für das Auslagern von Daten in den Erweiterungsspeicher geeignet ist.

(Heinrich Kiefl/rm)

```
[XY3]    1: /* Programm: extm_tst.c
[FC6]    2:    Funktion: extm_trb.sys Treiber testen
[9L2]    3:    Datum   : 23.Ø5.1991
[C93]    4:    Sprache : Turbo C 2.Ø
[SR3]    5:    Autor   : Heinrich Kiefl
[T15]    6:    (c)1991 DMV Widuch GmbH & Co. KG */
[ØØØ]    7:
[L13]    8: #include <stdio.h>
[RØ3]    9: #include <fcntl.h>
[VG2]   1Ø: #include <io.h>
[U83]   11: #include <stdlib.h>
[ØØØ]   12:
[RP2]   13: #define LAENGE 1ØØ
[ØØØ]   14:
[CL2]   15: char *buf_test =
[WH5]   16:   ". Satz im EXTM von C aus 1234567890";
[ØØØ]   17:
[C26]   18: char buf[LAENGE];  /* Arbeits Puffer */
[EØ2]   19: int handle;
[ØØØ]   2Ø:
```

```
[161]   21: struct
[4C3]   22: { char extm_adr[5];
[4L5]   23:   /* Adresse Erweiterungsspeicher */
[KS5]   24:   unsigned int len;  /* Statzlänge */
[RA4]   25:           int key;   /* Schlüssel */
[782]   26: }tabelle[5Ø];
[ØØØ]   27:
[UWØ]   28: main()
[F91]   29: { int i;
[394]   3Ø:   char datei[] = "EXTMMTRB";
[LF2]   31:    /* Treibername */
[EA6]   32:   handle = _open(datei,O_RDWR|O_TEXT);
[GB3]   33:   ioctl(handle,3,buf,1);
[VN4]   34:    /* Adresse von buf übergeben */
[YP2]   35:   for(i=Ø;i<=49;i++)
[UG2]   36:   { itoa(i,buf,1Ø);
[TN3]   37:     strcat(buf,buf_test);
[XU4]   38:     tabelle[i].len = strlen(buf);
[853]   39:     tabelle[i].key = i;
[XC5]   4Ø:     /* Satz in Erweiterungsspeicher
```

```
[6V1]   41:        schreiben */
[GØ6]   42:     ioctl(handle,3,tabelle[i].extm_adr,
[3N2]   43:           tabelle[i].len);
[U7Ø]   44:   }
[B71]   45:   lesen();
[W53]   46:   ioctl(handle,3,"A",1);
[W92]   47:   close(handle);
[U7Ø]   48: }
[ØØØ]   49:
[G51]   5Ø: lesen()
[WV6]   51:  /* Verschiedene Funktionen des Treibers */
[F91]   52: { int i;
[LD2]   53:   strcpy(buf,"");
[MA4]   54:   /* Datensätze auflisten */
[YP2]   55:   for(i=Ø;i<=49;i++)
[NE6]   56:   { ioctl(handle,2,tabelle[i].extm_adr,
[3N2]   57:                        tabelle[i].len);
[NR1]   58:         puts(buf);
[U7Ø]   59:   }
[P74]   6Ø:   /* Datensatz auswählen  */
[PW4]   61:   for(i=Ø;tabelle[i].key <= 3;i++)
[NE6]   62:   { ioctl(handle,2,tabelle[i].extm_adr,
[3N2]   63:           tabelle[i].len);
[ZM3]   64:     /* und neu schreiben      */
[U34]   65:     strcpy(buf,"Neuer Satz");
[GØ6]   66:     ioctl(handle,3,tabelle[i].extm_adr,
[3N2]   67:           tabelle[i].len);
[U7Ø]   68:   }
[G82]   69:   /* Auflisten */
[YP2]   7Ø:   for(i=Ø;i<=49;i++)
[DØ6]   71:     ioctl(handle,2,tabelle[i].extm_adr,
[3N2]   72:           tabelle[i].len);
[X14]   73:     /* Kennzeichen 'L' ändern */
[PW4]   74:   for(i=Ø;tabelle[i].key <= 3;i++)
[BS4]   75:     tabelle[i].extm_adr[Ø] = 'C';
[LD2]   76:   strcpy(buf,"");
[D82]   77:   /* Auflisten  */
[YP2]   78:   for(i=Ø;i<=49;i++)
[NE6]   79:   { ioctl(handle,2,tabelle[i].extm_adr,
[3N2]   8Ø:           tabelle[i].len);
[NR1]   81:     puts(buf);
[U7Ø]   82:   }
[U7Ø]   83: }
```

Listing 1. »extm__tst.c« ist ein Testprogramm für den Treiber

```
[S84]    1: ; Programm: extm_trb.asm
[WG5]    2: ; Fuktion: Datenübertragung vom
[AG4]    3: ;           Arbeitsspeicher in den
[2P4]    4: ;           Erweiterungsspeicher
[M13]    5: ;           und umgekehrt
[MX2]    6: ; Sprache: Tasm 1.Ø
[K34]    7: ; Autor    : Heinrich Kiefl
[FU4]    8: ; (c)1991 DMV Widuch GmbH & Co. KG
[P3Ø]    9: ;
[NG3]   1Ø: ; Treibererstellung:
[9F5]   11: ;  exe2bin extm_trb extm_trb.sys
[P3Ø]   12: ;
[FG3]   13: title EXT-mem Treiber
[NK1]   14: page 6Ø,132
[5Z1]   15: .MODEL TINY
[A16]   16: ; Konstanten (Adressen im Datenblock)
[9U1]   17: bef       equ 2
[F42]   18: anz       equ 18
[HG2]   19: bef_anz   equ 16
[Y52]   2Ø: status    equ 3
[4F2]   21: end_adr   equ 14
[M72]   22: p_adr     equ 14
[6B1]   23: ; Makros
[G57]   24: include  pushall.mac ;Makro:Register sich.
[MF7]   25: include  popall.mac  ;Makro:Register zurue.
[NN1]   26: ; Segmente
[T83]   27: code      segment  para
[2S6]   28: assume    cs:code,ds:code,es:code,ss:code
[AT4]   29:           org Øh     ;bei Treibern org Øh
[343]   3Ø: ; Kopf des Treibers
[7Z5]   31: dd -1   ;Zeiger auf nächsten Treiber
[GØ5]   32: dw 11ØØØØØØØØØØØØØØØb ;Attributwort
[R66]   33: dw offset strat       ;Strategy-Routine
[Y46]   34: dw offset inter       ;Interupt-Routine
[PU5]   35: db "EXTMMTRB"         ;Name des Treibers
[BX5]   36: ; Adresse der Parameter (Datenblock)
[8D6]   37: db_ptr   dd (Ø)         ;Daten-Block-Pointer
[2D3]   38: ; Funktions-Tabelle
[UE6]   39: fkt_tab  dw offset init    ;Funktion  Øh:
[PØ4]   4Ø: ;                  Initialisieren
[755]   41:          dw offset dummy   ;Funktion  1h:
[3D3]   42: ;                  Medium Test
[A55]   43:          dw offset dummy   ;Funktion  2h:
[HU3]   44: ;                   Erstellen BPD
[545]   45:          dw offset lesen   ;Funktion  3h:
[SZ3]   46: ;                  direktes lesen
[775]   47:          dw offset dir_no  ;Funktion  4h:
[7C2]   48: ;                  lesen
[525]   49:          dw offset besch   ;Funktion  5h:
[DM5]   5Ø: ;                  lesen, im Puffer bleiben
[825]   51:          dw offset besch   ;Funktion  6h:
```

```
[4W3]   52: ;                  Eingabe-Status
[T55]   53:          dw offset dummy   ;Funktion  7h:
[864]   54:          ; Eingabe Puffer löschen
[L75]   55:          dw offset dir_no  ;Funktion  8h:
[E23]   56: ;                  schreiben
[Z55]   57:          dw offset dummy   ;Funktion  9h:
[UM5]   58: ;                  schreiben und verändern
[R75]   59:          dw offset dummy   ;Funktion ØAh:
[7X3]   6Ø: ;                  Ausgabe-Status
[U75]   61:          dw offset dummy   ;Funktion ØBh:
[G65]   62: ;                  Ausgabe Puffer löschen
[WM5]   63:          dw offset schreiben ;Funktion ØCh:
[TT4]   64: ;                   direktes schreiben
[Ø85]   65:          dw offset dummy   ;Funktion ØDh:
[2F4]   66: ;                   öffnen (ab DOS 3.Ø)
[N45]   67:          dw offset besch   ;Funktion ØEh:
[G93]   68: ;                  schliessen
[685]   69:          dw offset dummy   ;Funktion ØFh:
[WU4]   7Ø: ;                  wechselbares Medium
[S25]   71:          dw offset besch   ;Funktion 1Øh:
[U75]   72: ;                  Ausgabe (beschäftigt)
[P3Ø]   73: ;
[593]   74: GDT            label word
[LA4]   75:                dw 4 dup (?)   ;Dummy Segment
[LP2]   76:                dw 4 dup (?)   ;GDT
[KA1]   77: ; Quelle
[2W6]   78: start_sg_len   dw ØFFFFh ;Segment l. Start
[1Z5]   79: start_lo       dw Ø      ;Quelle lo_word
[5X5]   8Ø: start_hi       db Ø      ;Quelle hi_byte
[UL3]   81:                db 93h    ;Zugriffs-Code
[Z24]   82:                dw Ø      ;reserviert für 386
[NZØ]   83: ; Ziel
[XK6]   84: ziel_sg_len    dw ØFFFFh ;Segment l. Ziel
[KA5]   85: ziel_lo        dw (Ø)    ;Ziel lo_word
[G75]   86: ziel_hi        db (Ø)    ;Ziel hi_byte
[UL3]   87:                db 93h    ;Zugriffs-Code
[Z24]   88:                dw Ø      ;reserviert für 386
[YXØ]   89: ; BIOS
[EP1]   9Ø:                dw 4 dup (?)
[D71]   91: ; Stack
[834]   92:                dw 4 dup (?);8 B f.Stack(GDT)
[UE1]   93: ; ENDE GDT
[P61]   94: ; DATEN
[EP5]   95: ram_buf        dw Ø,Ø ;Zeig. RAM-Puffer
[1G2]   96: richtung       db 2
[WT6]   97: ;Richtg:3 von Arbeits. nach Erweiterungs.
[Z36]   98: ;       2 von Erweiterungs.nach Arbeits.
[K67]   99: extm_total     dw Ø   ;Größe Erweiterungs.
[YFØ]  1ØØ:                db Ø
[ØM2]  1Ø1: start_adr      dw Ø
[RTØ]  1Ø2:                db 1Øh
[7T4]  1Ø3: ;Startadr. des Erweiterungs.
[NN2]  1Ø4: extm_frei      dw Ø
[YFØ]  1Ø5:                db Ø
[BD3]  1Ø6: ;Anzahl freier Bytes
[GD2]  1Ø7: anf_adr        dw Ø
[YFØ]  1Ø8:                db Ø
[A82]  1Ø9: ;ab hier Daten
[N66]  11Ø: satz_len       dw 256 ;Satzlänge in Byte
[2F6]  111: satz_len_w     dw 128 ;Satzlänge in Word
[GL2]  112: z_start        dw Ø
[YFØ]  113:                db Ø
[YE5]  114: ;Zwischenspeicher Startadresse
[WØ3]  115: z_extm_frei    dw Ø
[YFØ]  116:                db Ø
[KK6]  117: ;Zwischenspeicher freie Erweiterungs.
[P3Ø]  118: ;
[572]  119: ; Fehler Daten
[BX4]  12Ø: voll_f db   "@ EXTMEMORY VOLL",Ø
[Y33]  121: ; Strategy Routine
[AB2]  122: strat proc far
[7Ø4]  123:       mov word ptr db_ptr,bx
[964]  124:       mov word ptr db_ptr+2,es
[3PØ]  125:       ret
[DS1]  126: strat endp
[CH3]  127: ; Interrupt Routine
[CA2]  128: inter proc far
[VR5]  129:       PUSHALL  ;Makro: Register sichern
[CC4]  13Ø:       pushf    ;auch Flags sichern
[8C1]  131:       push cs
[5A1]  132:       pop  ds
[DP5]  133:       les  di,dword ptr db_ptr ;Adr. Dat.
[EY1]  134:                                ;Block holen
[1M5]  135:       mov  bl,es:[di+bef]  ;Befehl holen
[YX4]  136:       cmp  bl,bef_anz      ;Befehl o.k.
[3Z1]  137:       jle  bef_ok
[CM4]  138:       mov  ax,8ØØ3h        ;Fehler Code
[R33]  139:       jmp  short inter_e
[H91]  14Ø: bef_ok:
[KC1]  141:       shl  bl,1
[LN1]  142:       xor  bh,bh
[EZ2]  143:       call [fkt_tab+bx]
[C34]  144:       les  di,dword ptr db_ptr
[ØH1]  145: inter_e:
[1X4]  146:        or   ax,Ø1ØØh  ;Fertig Bit setzen
[C14]  147:        mov  es:[di+status],ax
[LSØ]  148:        popf
```

Listing 2. »extm__trb.asm« ist der Treiber für den Erweiterungsspeicher

```
[9L5]  149:         POPALL      ;Register zurückholen
[3PØ]  15Ø:         ret
[ØØØ]  151:
[2W1]  152: inter   endp
[E57]  153: ; Dummy Routine: Wird nur von Blocktreiber
[GS2]  154: ;                  benutzt
[W47]  155: dummy   proc near ;nichtgebrauchte Routinen
[YS1]  156:         xor  ax,ax
[3PØ]  157:         ret
[7W1]  158: dummy   endp
[WM5]  159: ; Für falsch aufgerufene Routinen
[BV2]  16Ø: dir_no  proc near
[2V5]  161:         mov ax,8ØØ3 ;Fehler:falscher Befehl
[3PØ]  162:         ret
[7Ø2]  163: dir_no  endp
[X14]  164: ; Beschäftigt Bit setzen
[AN2]  165: besch   proc near
[5V1]  166:         mov ax,Ø2ØØh
[3PØ]  167:         ret
[6U1]  168: besch   endp
[N84]  169: ; Routine 3: lesen d.h. von
[RL7]  17Ø: ; Erweiterungsspeicher nach Arbeitsspeicher
[CP2]  171: lesen   proc near
[EC1]  172:         push es
[RF5]  173:         les  bx,es:[di+p_adr] ; prüfe 'L'
[DY3]  174:         cmp  byte ptr es:[bx],'L'
[8A1]  175:         pop  es
[622]  176:         jne  les_end
[8E3]  177:         mov  cx,es:[di+anz]
[P75]  178: ; Anzahl Zeichen aaus Datenblock
[B52]  179:         jcxz les_end
[HF2]  18Ø:         test cx,ØØØØØØØ1b
[9K4]  181: ; Test auf gerade Satzlänge
[ZK1]  182:         jz   ger_1
[3V4]  183:         add  cx,1  ;nein: gerade machen
[A21]  184: ger_1:
[UU2]  185:         mov  satz_len,cx
[L24]  186: ;Satzlänge abspeichern
[XZ1]  187:         call teilen
[L12]  188:         call les_adr
[4N2]  189:         mov  richtung,2
[E42]  19Ø:         call ext_mem
[MØ2]  191:         call adr_ber
[GT4]  192:         mov  di,satz_len ;Ø anhängen
[NY4]  193:         les  bx,dword ptr [ram_buf]
[P84]  194:         mov  byte ptr es:[bx+di],Ø
[GE1]  195: les_end:
[2K4]  196:         xor ax,ax  ;Status-Byte o.k.
[3PØ]  197:         ret
[8V1]  198: lesen   endp
[MD7]  199: ; Routine Ø8: schreiben vom Arbeitsspeicher
[8Y5]  2ØØ: ;             in den Erweiterungsspeicher
[BH3]  2Ø1: schreiben     proc near
[8E3]  2Ø2:         mov  cx,es:[di+anz]
[993]  2Ø3:         ;Satzlänge nach CX
[4E1]  2Ø4:         cmp  cx,1
[UX3]  2Ø5:         jbe  buffer      ; Ø oder 1 ?
[K14]  2Ø6:         test cx,ØØØØØØØ1b ;gerade ?
[ML1]  2Ø7:         jz   ger_s
[2X4]  2Ø8:         add  cx,1   ; nein: gerade machen
[S21]  2Ø9: ger_s:
[RW2]  21Ø:         mov satz_len,cx
[UE5]  211:         call frei  ;freien Platz berechnen
[4M2]  212:         cmp  satz_len,Ø
[9S1]  213:         jne  sp_ok
[EM4]  214:         mov  ax,1ØØØ1Ø1Øb ;Status-Byte
[3PØ]  215:         ret
[551]  216: sp_ok:
[XZ1]  217:         call teilen
[GN4]  218:         mov richtung,3 ; 3=Ram -> EXTM
[E42]  219:         call ext_mem
[EC1]  22Ø:         push es
[7H3]  221:         les  bx,es:[di+p_adr]
[V44]  222:         ;Startadresse berechnen
[DY3]  223:         cmp  byte ptr es:[bx],'L'
[8A1]  224:         pop  es
[2Ø2]  225:         jne  ber_no
[MØ2]  226:         call adr_ber
[5W2]  227:         jmp  short schr_e
[BB1]  228: ber_no:
[T85]  229:         call st_adr ;neue Startadresse
[5Y2]  23Ø:         jmp short schr_e
[9A1]  231: buffer:
[7H3]  232:         les  bx,es:[di+p_adr]
[2Y3]  233:         cmp  byte ptr es:[bx],'A'
[N12]  234:         jne  buf_adr
[F52]  235:         call kennz_a
[5W2]  236:         jmp  short schr_e
[AE1]  237: buf_adr:
[D35]  238:         mov  cs:ram_buf+2,es ; Seg-Offs
[UU5]  239:         mov  cs:ram_buf,bx   ; nach ram_buf
[PA1]  24Ø: schr_e:
[2L3]  241:         xor ax,ax  ;Arbeit o.k.
[3PØ]  242:         ret
[9N2]  243: schreiben     endp
[E87]  244: ; Update: Startadresse nach z_start u. Adr.
[BZ5]  245: ; aus Kennzeichen 'L' in Startadresse
[UT2]  246: update  proc near
[CT4]  247:         mov  ax,word ptr start_adr
[ØZ4]  248:         mov  dl,byte ptr start_adr+2
[T44]  249:         mov  word ptr z_start,ax
[7C4]  25Ø:         mov  byte ptr z_start+2,dl
[YH4]  251:         mov  ax,word ptr es:[bx]+1
[UG4]  252:         mov  dl,byte ptr es:[bx]+3
[5L4]  253:         mov  word ptr start_adr,ax
[BS4]  254:         mov  byte ptr start_adr+2,dl
[3PØ]  255:         ret
[PY1]  256: update  endp
[524]  257: ; Startadr. berichtigen
[8Z2]  258: adr_ber proc near
[BD4]  259:         mov  ax,word ptr z_start
[DL4]  26Ø:         mov  dl,byte ptr z_start+2
[5L4]  261:         mov  word ptr start_adr,ax
[BS4]  262:         mov  byte ptr start_adr+2,dl
[3PØ]  263:         ret
[442]  264: adr_ber endp
[ØK4]  265: ; Freien Speicher berechnen
[A83]  266: ;  u. auf voll prüfen
[RK2]  267: frei    proc near
[TN4]  268:         PUSHALL     ;Register sichern
[7H3]  269:         les  bx,es:[di+p_adr]
[DY3]  27Ø:         cmp  byte ptr es:[bx],'L'
[C81]  271:         jne  fr
[UØ2]  272:         call update
[YT3]  273:         jmp  short sp_voll_no
[ALØ]  274: fr:
[GV2]  275:         mov ax,satz_len
[TH1]  276:         xor dx,dx
[YK5]  277:         mov bx,extm_frei ;Noch freie Byte
[AV4]  278:         mov dl,byte ptr extm_frei+2
[UF1]  279:         sub bx,ax
[Ø71]  28Ø:         sbb dl,Ø
[RØ2]  281:         jc  sp_voll
[1Y2]  282:         mov extm_frei,bx
[GV4]  283:         mov byte ptr extm_frei+2,dl
[FS3]  284:         jmp short sp_voll_no
[MK1]  285: sp_voll:
[ZW4]  286:         les  di,dword ptr [ram_buf]
[DX5]  287:         lea  si,voll_f ;Offset Fehler Zeile
[DS1]  288:         mov  ax,cs
[DS1]  289:         mov  ds,ax
[6H4]  29Ø:         mov  cx,17      ;Anzahl Zeichen
[U83]  291:         cld             ;aufwärts
[SR1]  292:   rep   movsb
[PN2]  293:         mov  satz_len,Ø
[4R5]  294:         ;Satzlänge=Ø für Rücksprung
[265]  295:         mov  ah,Ø2h     ;Pip-Ton erzeugen
[CL1]  296:         mov  dl,Ø7
[381]  297:         int 21h
[A72]  298: sp_voll_no:
[N46]  299:         POPALL  ;Register wieder herstellen
[3PØ]  3ØØ:         ret
[CR1]  3Ø1: frei    endp
[Z36]  3Ø2: ; Satzlänge durch 2 teilen für Words
[9N2]  3Ø3: teilen proc near
[GV2]  3Ø4:         mov ax,satz_len
[TH1]  3Ø5:         xor dx,dx
[FB1]  3Ø6:         mov cx,2
[931]  3Ø7:         div cx
[CD1]  3Ø8:         add ax,dx
[U33]  3Ø9:         mov satz_len_w,ax
[3PØ]  31Ø:         ret
[8Y1]  311: teilen endp
[C55]  312: ; Startadr. in Tabelle schreiben
[NW4]  313: ; und neue Startadr. berechnen
[BT2]  314: st_adr proc near
[CT4]  315:         mov  ax,word ptr start_adr
[ØZ4]  316:         mov  dl,byte ptr start_adr+2
[7H3]  317:         les  bx,es:[di+p_adr]
[ZZ3]  318:         mov  byte ptr es:[bx],'L'
[LA4]  319:         mov  word ptr es:[bx]+1,ax
[A94]  32Ø:         mov  byte ptr es:[bx]+3,dl
[214]  321:         mov  byte ptr es:[bx]+4,Ø
[626]  322:         mov ax,satz_len ;neue Startadresse
[7S2]  323:         add start_adr,ax
[VE4]  324:         adc byte ptr start_adr+2,Ø
[3PØ]  325:         ret
[A22]  326: st_adr endp
[2M6]  327: ; Startadr. von Anwenderprogramm laden
[ØØ3]  328: les_adr proc near
[CT4]  329:         mov  ax,word ptr start_adr
[4C2]  33Ø:         ;Startadresse
[ØZ4]  331:         mov  dl,byte ptr start_adr+2
[T44]  332:         mov  word ptr z_start,ax
[GK3]  333:         ;in Zwischenspeicher
[7C4]  334:         mov  byte ptr z_start+2,dl
[7H3]  335:         les  bx,es:[di+p_adr]
[YH4]  336:         mov  ax,word ptr es:[bx]+1
[UG4]  337:         mov  dl,byte ptr es:[bx]+3
[5L4]  338:         mov  word ptr start_adr,ax
[BS4]  339:         mov  byte ptr start_adr+2,dl
[3PØ]  34Ø:         ret
[L72]  341: les_adr  endp
[RS5]  342: ; Anfangsadr. wieder herstellen: 'A'
[LD3]  343: ; und freien Speicher
[243]  344: kennz_a  proc near
[B74]  345:         mov  ax,word ptr anf_adr
[DD4]  346:         mov  dl,byte ptr anf_adr+2
[5L4]  347:         mov  word ptr start_adr,ax
[BS4]  348:         mov  byte ptr start_adr+2,dl
[M55]  349:         mov  ax,word ptr z_extm_frei
[PB5]  35Ø:         mov  dl,byte ptr z_extm_frei+2
```

```
[WK4]  351:         mov  word ptr extm_frei,ax
[2S4]  352:         mov  byte ptr extm_frei+2,dl
[3PØ]  353:         ret
[B82]  354: kennz_a  endp
[8V3]  355: ; Befehle zum GDT aufruf
[9R4]  356: ext_mem  proc near ;Proc GDT
[TE3]  357:          ; Speicherberechnung
[9S4]  358:          PUSHALL   ;Register sichern
[1G4]  359:          mov ax,word ptr start_adr
[NT4]  36Ø:          mov dl,byte ptr start_adr+2
[MD2]  361:          cmp richtung,2
[DZ3]  362:          jne richt_3 ;Richtung 2
[KX5]  363:          mov start_hi,dl ;EXTM nach Hi_Byte
[226]  364:          mov start_lo,ax ;EXTM nach Lo_Word
[GE5]  365:          mov ax,[ram_buf+2] ;Seg ram_buf
[XW4]  366:          mov bx,[ram_buf]   ;Offset dto
[7A5]  367:          call adresse ;24bit Adr. berechn.
[NM5]  368:          mov  ziel_hi,dl ;RAM nach Hi_Byte
[VR5]  369:          mov  ziel_lo,ax ;RAM nach Lo_Word
[GP3]  37Ø:          jmp  short gdt_aufruf
[816]  371: richt_3:       ;Richtung 3 von RAM -> EXTM
[1V5]  372:          mov  ziel_hi,dl ;EXTM nach Hi_Byte
[HZ5]  373:          mov  ziel_lo,ax ;EXTM nach Lo_Word
[SM5]  374:          mov  ax,[ram_buf+2] ;Seg ram_buf
[CE5]  375:          mov  bx,[ram_buf]   ;Offset dto
[Y12]  376:          call adresse
[6R5]  377:          mov start_hi,dl ;RAM nach Hi_Byte
[CU5]  378:          mov start_lo,ax ;RAM nach Lo_Word
[S22]  379: gdt_aufruf:
[F84]  38Ø:          mov ah,Ø87h ;Fkt 87h nach ah
[X25]  381:          mov cx,satz_len_w ;Anz. Words
[8C1]  382:          push cs
[G31]  383:          pop es
[YV2]  384:          mov si,offset GDT
[D81]  385:          int 15h
[5M3]  386:          jmp short extm_raus
[4G6]  387:    adresse   proc near ;Proc 24bit Adr-Ber.
[CF1]  388:              mov dl,ah
[9M1]  389:              mov cl,Ø4h
[XE1]  39Ø:              shr dl,cl
[3F1]  391:              shl ax,cl
[AD1]  392:              add ax,bx
[T92]  393:              jnc k_ueb_tr
[MØ1]  394:              inc dl
[ØD6]  395:  k_ueb_tr:   ret    ;zur. in Proc ext_mem
[VD2]  396:  adresse     endp
[GW1]  397: extm_raus:
[EA5]  398:          POPALL  ;Register wieder herst.
[3PØ]  399:          ret
[992]  4ØØ: ext_mem  endp
[VN2]  4Ø1: ; Ende GDT Befehle
[LM4]  4Ø2: ; Initialisierungs-Routine
[CL2]  4Ø3: init     proc near
[BC1]  4Ø4:          push ds
[EX2]  4Ø5:          mov ax,Øf ØØØh ; AT ?
[1H1]  4Ø6:          mov ds,ax
[9G4]  4Ø7:          test byte ptr ds:[Øfffeh],3
[KA1]  4Ø8:          je at_ja
[E31]  4Ø9:          pop ds
[4Z4]  41Ø:          mov dx,offset kein_at ;Fehler
[6K4]  411:          mov ah,Ø9h     ;Meldung kein AT
[381]  412:          int 21h
[DV1]  413:          mov ax,8ØØ1h
[3PØ]  414:          ret
[M31]  415: at_ja:
[E31]  416:          pop ds
[UE5]  417:          mov dx,offset meld ;Init. Meldung
[3M1]  418:          mov ah,Ø9h
[381]  419:          int 21h
[D13]  42Ø:          mov  ax,offset buf
[ZS2]  421:          mov  ram_buf,ax
[Ø14]  422:          mov  ram_buf+2,cs ;   info
[E42]  423:          call ext_mem
[ZH2]  424:          call extm_init
[G57]  425:   mov word ptr es:[di+end_adr],offset init
[R14]  426:          mov es:[di+end_adr+2],cs
[GH1]  427:          xor ax,ax
[3PØ]  428:          ret
[8S1]  429: init     endp
[E92]  43Ø: ; Treiber init
[LW6]  431: ; Namen vergleichen ob RAM-Disk vorhanden
[ZG7]  432: ; wenn ja: Größe der RAM-Disk zu Startadr.
[VD6]  433: ;      nein: ExtMem Control-Block anlegen
[DD3]  434: extm_init proc near
[HG4]  435:           PUSHALL  ;Register sichern
[9M1]  436:           mov ah,88h
[NB4]  437:           ;EXTM Groesse feststellen
[P64]  438:           int 15h    ;BIOS Interrupt
[F25]  439:           mov cx,1Ø24 ; in Byte umrechnen
[R31]  44Ø:           mul cx
[2V4]  441:           mov word ptr extm_total,ax
[SØ5]  442:           mov byte ptr extm_total+2,dl
[2H1]  443:           mov ax,cs
[3H1]  444:           mov es,ax
[PS2]  445:           lea di,buf_vergl
[1H1]  446:           mov ds,ax
[UL1]  447:           lea si,buf
[KE1]  448:           mov cx,31
[W54]  449:           ;mit den ersten 31 Zeichen
[UY2]  45Ø:           cld          ;aufwärts
[614]  451:       repe cmpsb       ;byteweise
[1E5]  452:           jne nam_no ;nicht->keine RAM-Disk
```

```
[LC4]  453:           jmp short nam_y ;RAM-Disk
[3B1]  454: nam_no:
[ZU2]  455:           lea ax,extm_ctrl
[2H4]  456:           ;Adr. eigener Control-Block
[965]  457:           mov ram_buf,ax   ; nach ram_buf
[6S2]  458:           mov ram_buf+2,cs
[9F2]  459:           mov richtung,3
[E42]  46Ø:           call ext_mem
[GV2]  461:           mov ax,satz_len
[7S2]  462:           add start_adr,ax
[VE4]  463:           adc byte ptr start_adr+2,Ø
[6F4]  464:           ;Freien Speicher berechnen
[YK4]  465:           xor dx,dx          ;keine Angaben
[ØR5]  466:           mov bx,extm_total ;über RAM-Disk
[ØØ5]  467:           mov dl,byte ptr extm_total+2
[TF5]  468:           sub bx,ax ;minus AX (Satzlänge)
[Ø71]  469:           sbb dl,Ø
[MR4]  47Ø:           mov extm_frei,bx ;noch frei
[GV4]  471:           mov byte ptr extm_frei+2,dl
[WE3]  472:           mov z_extm_frei,bx
[U45]  473:           mov byte ptr z_extm_frei+2,dl
[9F1]  474:           jmp raus
[BØ4]  475: ; Felder initialisieren
[DD3]  476: ; RAM-Disk Info in buf
[U16]  477: ; Umrechnen in Byte und abspeichern
[K21]  478: nam_y:
[ZD4]  479:           mov cx,1Ø24 ;Multiplikator
[WS3]  48Ø:           mov ax,word ptr buf+7Ø
[MT3]  481:           mov bx,word ptr buf+68
[U65]  482:           mov start_adr,bx      ;Byte(long)
[3Y4]  483:           mov byte ptr start_adr+2,al
[2T3]  484:           mov ax,word ptr buf+72
[VA5]  485:           mul cx ;Kbyte umrechnen in Byte
[N45]  486:           add start_adr,ax ;zu Startadr.
[BU4]  487:           add byte ptr start_adr+2,dl
[YS3]  488:           mov ax,word ptr buf+52
[3E3]  489:           ;freie KByte im EXTM
[R31]  49Ø:           mul cx
[8B4]  491:           mov extm_frei,ax ;in Byte
[GV4]  492:           mov byte ptr extm_frei+2,dl
[5S4]  493:           mov z_extm_frei,ax ;merken
[U45]  494:           mov byte ptr z_extm_frei+2,dl
[2WØ]  495: raus:
[PV2]  496:           mov ax,start_adr
[NT4]  497:           mov dl,byte ptr start_adr+2
[EH2]  498:           mov anf_adr,ax
[H64]  499:           mov byte ptr anf_adr+2,dl
[5S3]  5ØØ:           POPALL ;Register laden
[3PØ]  5Ø1:           ret
[5P2]  5Ø2: extm_init   endp
[AZ1]  5Ø3: ; Meldungen
[LW2]  5Ø4: buf          db 256 dup (Ø)
[DK5]  5Ø5: buf_vergl    db 'MICROSOFT EMM CTRL '
[FK4]  5Ø6:              db 'VERSION 1.ØØ',Ø ;1 Z. z. vgl
[XE6]  5Ø7: extm_ctrl    db 'EXTENDED MEMORY TREIBER '
[VR2]  5Ø8:              db 'VERSION 1.Ø1  '
[5K2]  5Ø9:              db 'von H. Kiefl '
[HP1]  51Ø:              db 2ØØ dup (Ø)
[L95]  511: meld         db 13,1Ø,"Extended-Memory-"
[FX3]  512:              db  "Treiber Version 1.Ø1"
[VF3]  513:              db " installiert.",13,1Ø
[V34]  514:              db "(c)1991 DMV Widuch GmbH"
[CK1]  515:              db " & Co. KG"
[DA1]  516:              db 13,1Ø,"$"
[SU4]  517: kein_at      db "Ihr PC ist kein AT"
[DA1]  518:              db 13,1Ø,"$"
[6H2]  519: ; Programm Ende
[1R1]  52Ø: code    ends
[4MØ]  521:         end
```

Listing 2. »extm_trb.asm« ist der Treiber für den Erweiterungsspeicher

```
[632]    1: ;pushall.mac
[2A2]    2: PUSHALL MACRO
[TB1]    3:   push bp
[7S1]    4:   mov  bp,sp
[WN5]    5:   IRP  REG, <ax,bx,cx,dx,si,di,ds,es>
[4E1]    6:   push REG
[HPØ]    7: ENDM
[HPØ]    8: ENDM
[ØØØ]    9:
```

Listing 3. »pushall.mac« enthält ein Makro zum Sichern der Prozessor-Register

```
[ZØ2]    1: ;popall.mac
[T12]    2: POPALL MACRO
[KG5]    3:   IRP REG, <es,ds,di,si,dx,cx,bx,ax>
[WB1]    4:   pop REG
[HPØ]    5: ENDM
[131]    6:   pop bp
[HPØ]    7: ENDM
```

Listing 4. »popall.mac« stellt ein Makro zum Laden der Prozessor-Register bereit

GEM-Programmierung

Edelstein für Turbo Pascal

Die grafische Benutzeroberfläche GEM hat sich einen festen Platz in der Welt der PCs gesichert, besonders im Bereich der Präsentationsgrafik und des Desktop Publishing. Erfahren Sie hier, wie Sie selbst GEM-Programme in Turbo Pascal entwickeln.

Der Unterschied zwischen MS-DOS-Welt und Atari-Welt besteht darin, daß Sie auf PCs meist nur GEM-Programme von kommerziellen Anbietern bekommen, während die GEM-Programmierung auf dem Atari zum Standard der Public-Domain-Programme gehört. Einer der Gründe hierfür ist sicherlich, daß fast jeder Compiler für den Atari eine GEM-Bibliothek enthält und Sie diese auf dem MS-DOS-PC erst teuer nachkaufen müssen, falls Sie sie überhaupt für Ihre bevorzugte Programmiersprache bekommen.

Wenn Sie GEM-Programmierung bereits vom Atari ST her kennen und auf dem PC vermissen, vielleicht sogar Programme auf den PC übertragen wollen, dann können Sie aufatmen. Wenn Sie kein Vermögen für das GEM-Programmers-Toolkit ausgeben wollen und die Programmentwicklung unter Turbo Pascal ablaufen soll, bietet Ihnen die nachfolgend beschriebene Unit »gembind.tpu« (Tabelle 1) eine bequeme Alternative (Listing 1). Die Funktionsaufrufe sind kompatibel zur GEM-Bibliothek des ST-Pascal Plus, dem am weitesten verbreiteten Pascal auf dem ST.

Die einzige Voraussetzung, um mit der Unit GEM-Programme schreiben zu können, ist der Besitz von GEM-Desktop, das jeder GEM-Anwendung beiliegt. Dieses beinhaltet »gemvdi.exe«, den residenten Teil von GEM. Der Name ist allerdings irreführend, da darin sowohl das AES als auch das VDI enthalten sind.

AES und VDI sind die wichtigsten Komponenten

GEM (englisch: Edelstein), die Abkürzung für Graphics Environment Manager, besteht im wesentlichen aus zwei Teilen: dem AES und dem VDI. AES steht für Application Environment Services; es enthält etliche Funktionen für eine standardisierte Kommunikation mit dem Benutzer, so zum Beispiel Alert-Boxen zum Anzeigen von kurzen Meldungen, Eingabefenster (Dialoge), Drop-down-Menüs und Fenster. Diese Objekte verwaltet das AES intern als Bäume mit vielen Teilobjekten.

Das VDI (Virtual Device Interface) ist eine Sammlung von geräteunabhängigen Grafikfunktionen. Mit ihnen produzieren Sie den Inhalt eines Fensters, während das AES für die Fensterverwaltung (Verschieben, Größe verändern und so weiter) zuständig ist. Die geräteunabhängige Komponente des VDI erlaubt es zum Beispiel, daß nach der Installation einer neuen Grafikkarte mit dem entsprechenden GEM-Treiber sofort alle GEM-Programme die neuen Fähigkeiten ausnutzen können, ohne daß Sie sie speziell anpassen müssen.

Das Herzstück der GEM-Bibliothek »gembind.tpu« sind die Prozeduren »AES_Call« und »VDI_Call«. Sie stellen einen Parameterblock mit den Adressen der Variablenfelder für GEM zusammen und legen die Adresse dieses Parameterblocks in den Registern »ES:BX« ab. Im Register »CX« legt »gembind.tpu« eine Kennzahl ab, die angibt, ob es sich um einen AES- oder VDI-Aufruf handelt. Für einen AES-Aufruf ist das 200, bei einem VDI-Aufruf 1139. Nun löst das Programm den Interrupt EFhex aus, der GEM aufruft. Die Anzahl der Ein- und Ausgabeparameter für die AES-Funktionen stehen in der typisierten Konstante »CRYSTAL«. Bei VDI-Funktionen übergibt sie das Programm immer direkt. Diese unterschiedliche Behandlung von AES- und VDI-Funktionen ist notwendig, um die Kompatibilität mit ST-Pascal Plus zu wahren.

Die Belegung der Parameterblöcke und der Variablenfelder für die einzelnen GEM-Funktionen können Sie der Dokumentation zum GEM-Programmers-Toolkit entnehmen. Demjenigen, dessen Brieftasche nicht dick genug ist, diese Original-Dokumentation zu erstehen, sei das Atari-ST-Profibuch empfohlen, dessen Softwareteil zu

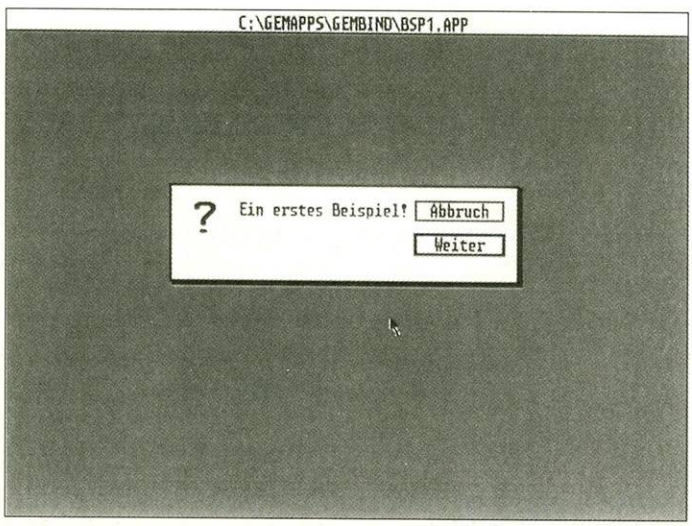


Bild 1. Das Programm »alert.pas« zeigt diese Alert-Box

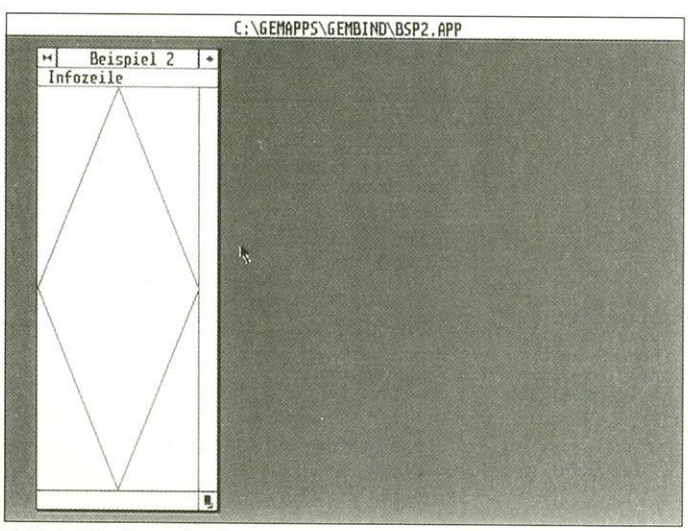


Bild 2. Die Ausgabe des Programms »fenster.pas«

	Befehl Art	Bedeutung
Grafikfunktionen	Draw_Mode	Schreibmodus festlegen (überschreiben, transparent etc.)
	Draw_String	einen Text ausgeben
	Frame_Rect	Rahmen eines Rechtecks zeichnen
	Growbox	ein sich vergrößerndes Rechteck animieren (nur in GEM 1.0)
	Line_Color	Zeichenfarbe für Linien auswählen
	Line_Endstyle	Art der Linienenden festlegen (rund, eckig, Pfeil)
	Line_Style	Linienmuster auswählen (durchgehend, unterbrochen etc.)
	Paint_Color	Zeichenfarbe für ausgefüllte Flächen festlegen
	Paint_Style	Muster für gefüllte Flächen festlegen
	Paint_Rect	gefülltes Rechteck zeichnen
	Pline	Linienzug zeichnen
	Set_Clip	Grafikausgabe auf ein bestimmtes Rechteck begrenzen
	Shrinkbox	ein sich verkleinerndes Rechteck animieren (nur in GEM 1.0)
	Text_Height	Textgröße auswählen
	Text_Rotation	Rotationswinkel des Textes angeben (in 90-Grad-Schritten)
	Text_Style	Textattribute festlegen
Maussteuerung	Hide_Mouse	Mauszeiger verstecken
	Init_Mouse	Maussteuerung beim Programmstart initialisieren
	Set_Mouse	dem Mauszeiger eine andere Form geben
	Show_Mouse	Mauszeiger sichtbar machen
Fensterverwaltung	Border_Rect	Größe des Fensters abfragen (Fenstergröße außen)
	Close_Window	Fenster auf dem Bildschirm schließen
	Delete_Window	Datenstruktur des Fensters aus dem Speicher entfernen
	Front_Window	Fenster zum obersten machen
	New_Window	Datenstruktur für ein neues Fenster erzeugen
	Open_Window	ein Fenster auf dem Bildschirm öffnen
	Set_WSize	Fenstergröße ändern
	Set_Winfo	Text der Infozeile des Fensters setzen
	Set_WName	Titelzeile des Fensters ändern
	Set_Window	aktiviert das angegebene Fenster (das aktive Fenster ist nicht zwingend das oberste)
	Wind_Get	liefert je nach Argument verschiedene Informationen über ein Fenster (zum Beispiel über die Rollbalken)
	Wind_Set	verändert je nach Argument verschiedene Teile des Fensters (zum Beispiel die Rollbalken)
	Work_Rect	Größe des Arbeitsbereichs abfragen (Fenstergröße innen)
Eventverwaltung	First_Rect	ersten neu zu zeichnenden Bildschirmbereich (Rechteck) erfragen
	Get_Event	Meldungen (Events) von GEM abwarten
	Next_Rect	nächsten neu zu zeichnenden Bildschirmbereich erfragen
	Rect_Intersect	feststellen, ob sich zwei Rechtecke überlappen
Dialogverwaltung	Center_Dialog	den Dialog auf der aktuellen Bidschirmauflösung zentrieren
	Do_Dialog	Dialog darstellen und Benutzer-Interaktion zulassen
	End_Dialog	Dialog vom Bildschirm entfernen (Redraw-Mitteilung erzeugen)
	Get_DEdit	editierten Text aus Dialogbox auswerten
	Objc_Draw	ein einzelnes Objekt zeichnen
	Obj_SetState	Status eines Dialogobjekts (zum Beispiel Selektierung) setzen
	Obj_State	Status eines Objekts erfragen
	Show_Dialog	Dialog auf dem Bildschirm darstellen
	Set_DText	einem Textobjekt einen neuen Text zuweisen
Menüverwaltung	Draw_Menu	Menü auf dem Bildschirm darstellen
	Erase_Menu	Menü deaktivieren (bleibt sichtbar, bis ein neues installiert wird)
	Menu_Disable	Menüeintrag deaktivieren (helle Schrift)
	Menu_Enable	Menüeintrag aktivieren (normale Schrift)
	Menu_Hilight	Menütitel invertieren
	Menu_Normal	Menütitel normal darstellen
Ressourcenverwaltung	Find_Dialog	Handle eines bestimmten Dialogs aus der Ressource erfragen
	Find_Menu	Handle eines bestimmten Menüs aus der Ressource erfragen
	Free_Resource	Speicherplatz der Ressource am Programmende freigeben
	Load_Resource	Ressource laden
Diverses	AES_Call	AES-Aufruf direkt ausführen
	Begin_Update	Benutzeraktionen für die Zeit des Bildschirmaufbaus unterbinden. Diese Funktion wenden Sie vor dem Neuzeichnen eines Fensters an.
	Do_Alert	Alertbox entsprechend der übergebenen Formatzeichenkette darstellen
	Dragbox	bei gedrückter Maustaste den Benutzer ein Rechteck verschieben lassen
	End_Update	Benutzeraktionen wieder zulassen. Nach dem Sperren des Computers mit »Begin_Update« gibt diese Funktion ihn wieder für Benutzeraktionen frei.
	Exit_Gem	Anwenderprogramm bei GEM abmelden
	Get_In_File	Dateiauswahlbox öffnen
	Init_Gem	Anwenderprogramm bei GEM anmelden
	VDI_Call	VDI-Aufruf direkt ausführen

Tabelle 1. Sämtliche Befehle, die in der Unit »gembind.tpu« realisiert sind, sind nach Themengebieten geordnet. Nicht aufgeführte Kommandos können Sie leicht ergänzen.

GEM größtenteils eine Übersetzung des GEM-Programmers-Toolkit ist. Die dort zu findenden Informationen gelten auch für GEM für MS-DOS-PCs. Es sind nur wenige kleine Unterschiede zu beachten, die später näher betrachtet werden. Diese Detailkenntnisse sind allerdings nur dann nötig, wenn Sie die Bibliothek selbst erweitern möchten.

Um die GEM-Bibliothek in Turbo-Pascal-Programmen verwenden zu können, müssen Sie sie mit dem Befehl

```
uses gembind;
```

einbinden. Diese Unit ersetzt die Dateien »gemconst.pas«, »gemtype.pas« sowie »gemsubs.pas« des ST-Pascal Plus. Sofern Sie nicht mit GEM/2 oder mit GEM/3 auf dem PC arbeiten, ist die Programmierung fast identisch mit der auf dem Atari. In den beiden neueren Versionen von GEM sind aber nur wenige zusätzliche Funktionen hinzugekommen. Einige Besonderheiten sollten Sie jedoch bei der Verwendung von Turbo Pascal beachten. Damit die GEM-Programme getestet werden können, müssen sie als Exe-Datei auf der Festplatte vorliegen. Selbst wenn Sie Turbo Pascal von GEM aus aufgerufen haben, können sie GEM-Applikationen nur außerhalb von Turbo Pascal starten, da MS-DOS-Programme GEM abschalten und von ihnen aus keine GEM-Programme aufgerufen werden können. Der integrierte Debugger von Turbo Pascal ab Version 5.0 ist daher nicht zu gebrauchen.

Um der GEM-Bibliothek genügend Speicher im Heap für die eigene Arbeit zur Verfügung zu stellen, müssen Sie Turbo Pascal mit der Direktive »$M« anweisen, nicht den gesamten Speicher zu belegen. Eine brauchbare Standardeinstellung ist

```
(*$M 16384, 0, 10000*)
```

Sie haben zwei Optionen, ein selbstgeschriebenes GEM-Programm als solches starten können. Entweder Sie benennen die Dateierweiterung von »exe« in »app« um, um das Programm von GEM-Desktop aus zu starten. Oder Sie melden die Exe-Datei als GEM-Programm im Desktop an. Dazu selektieren Sie die Datei mit einem Mausklick und rufen den Menüpunkt »Anwendung konfigurieren« im Menü »Optionen« auf oder verwenden die Kurzwahltaste [Alt-A]. Sie müssen dem Desktop also mitteilen, daß es sich nicht um ein normales MS-DOS-Programm handelt. Sie sollten danach das Speichern der Einstellung im Desktop nicht vergessen.

Falls ein GEM-Programm mit einem Laufzeitfehler abbricht, ist die Fehlermeldung von Turbo Pascal meist nicht zu sehen. Manchmal blinkt sie kurz auf. Zu diesem Problem gibt es bisher keine Lösung. Alle Versuche es zu beheben, zum Beispiel mit einer Umleitung der Ausgabe, schlugen fehl. Hier ist Ihre Experimentierfreudigkeit gefragt.

Ein kurzes Programm läßt eine Alert-Box auf dem Bildschirm erscheinen

Genug der Vorrede: Sie können nun Ihr erstes GEM-Programm schreiben (Listing 2). Es initialisiert GEM und stellt eine Alert-Box dar (Bild 1). In Abhängigkeit von der angeklickten Schaltfläche zeigt es eine weitere Alert-Box. Wie Sie sehen, können Sie bereits mit wenigen Zeilen vollständige GEM-Programme erzeugen.

Am Anfang jedes Programms müssen Sie GEM mit »Init_Gem« initialisieren. Als Rückgabewert erhalten Sie die positive Application-ID, eine Kennziffer, die die gestartete Applikation eindeutig beschreibt, oder einen negativen Zahlenwert als Fehlermeldung. Beim Verlassen des Programms ist ein Abmelden mit »Exit_Gem« notwendig. Dies sollte allerdings nur nach gelungener Initialisierung erfolgen.

Auch die Alert-Boxen haben Sie bereits im ersten Beispielprogramm kennengelernt. Sie dienen dazu, kurze Mitteilungen oder Fragen auf den Bildschirm zu bringen. Die Formatzeichenkette, die Sie »Do_Alert« übergeben, hat die Form »[Symbol][Text][Schaltfläche]«. Tabelle 2 zeigt eine Übersicht der zugelassenen Symbole. Der Text kann bis zu fünf durch den ASCII-Code 124 getrennte Zeilen umfassen, die maximal 30 Zeichen lang sein dürfen. Weiterhin können Sie die Namen von bis zu drei Schaltflächen angeben, die Sie ebenfalls durch den ASCII-Code 124 trennen. Beim Aufruf von »Do_Alert« sichert GEM automatisch den Hintergrund und restauriert ihn nach Verlassen der Alert-Box. Nach dem Aufruf gibt »Do_Alert« die Nummer der vom Benutzer angeklickten Schaltfläche zurück, wobei 1 der ersten Schaltfläche in der Formatzeichenkette entspricht.

Dialoge verwenden Sie, um Eingabedaten zu erfragen, längere Mitteilungen auszugeben oder ähnliches. Sie werden gewöhnlich mit dem Resource Construction Set (RCS) erzeugt. Das ist ein Programm, mit dem Sie grafisch nach dem Baukastenprinzip verschiedene Grundobjekte zu einer Dialogbox oder einem Menü zusammenbauen und in einer Ressourcendatei speichern. Diese Ressourcendatei lädt dann das Programm nach. Dieses Verfahren ist für den Programmierer sehr angenehm, da er Programm und Benutzeroberfläche weitgehend unabhängig voneinander entwickeln kann. Kleine Änderungen an der Oberfläche sind durchführbar, ohne das Programm neu kompilieren zu müssen. Für Autoren ist diese Methode weni-

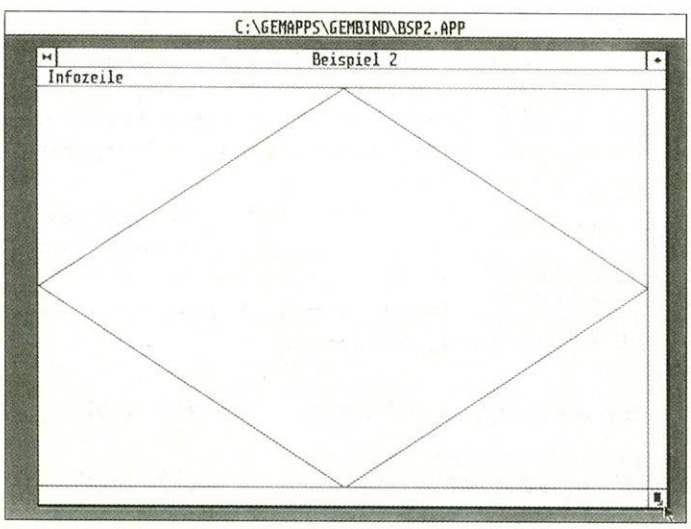


Bild 3. Auf Veränderungen der Fenstergröße reagiert »fenster.pas« mit der Aktualisierung des Fensterinhalts

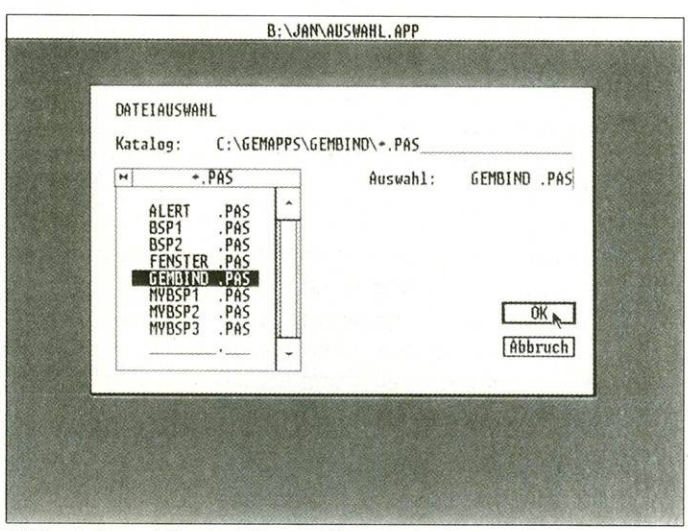


Bild 4. Eine solche Dateiauswahlbox erfordert nicht mehr als einen GEM-Befehl

ger angenehm, da man diese Ressourcen nicht einfach als Listing abdrucken kann. Daher sei kurz erwähnt, wie Sie mit Dialogen umgehen, ohne ein Beispiel dafür zeigen zu können. Zunächst laden Sie die Ressource mit »Load_Resource«. Danach suchen Sie mit »Find_Dialog« nach der gesuchten Dialogbox. Hierbei erhalten Sie einen Zeiger auf den zugehörigen Objektbaum. Die Prozedur »Center_Dialog« wird dann verwendet, um die Dialogbox auf dem aktuellen Bildschirm, der ja eine wechselnde Auflösung haben kann, zu zentrieren.

Bisher ist auf dem Bildschirm nichts zu sehen. Die Darstellung der Dialogbox erfolgt mit »Do_Dialog«, wobei Sie das erste zu editierende Textfeld als Parameter übergeben. Mit »Do_Dialog« editieren Sie die Dialogbox und erhalten anschließend die Nummer der Schaltfläche, über die der Benutzer den Dialog verlassen hat. Um den Dialog jetzt vom Bildschirm zu entfernen, müssen Sie »End_Dialog« aufrufen. Die Auswertung von Texteingaben erfolgt durch »Get_Dtext«. Sie übergeben den Zeiger auf den Dialog und die Nummer des editierten Objekts und erhalten den eingegebenen Text. Es ist wichtig, daß Sie vor dem erneuten Darstellen eines Dialogs zum Beispiel die zum Verlassen gewählte Schaltfläche mit »Obj_SetState« von »Selected« auf »Normal« setzen, sonst verläßt GEM den Dialog unmittelbar nach dem Aufbau. Bevor Sie das Programm beenden, sollten Sie mit »Free_Resource« den von der Ressource belegten Speicher freigeben.

Die verschiebbaren Fenster sind das Kernstück von GEM

Zum Entwurf eines Drop-down-Menüs wird gewöhnlich ebenfalls das Resource Construction Set verwendet. Auch hierzu folgt nun eine kurze Erklärung ohne Beispiel. Nach dem Laden der Ressource holen Sie sich mit »Find_Menu« einen Zeiger auf die Objektstruktur, die Sie dann mit »Draw_Menu« darstellen. Um dem Benutzer Gelegenheit zu geben, etwas aus dem Menü auszuwählen, rufen Sie in einer Schleife wiederholt »Get_Event« auf. Mit »Erase_Menu« müssen Sie das Menü vor dem Verlassen des Programms bei GEM abmelden.

Fenster sind die geeignete Darstellungsart für alle Objekte, die größer als der Bildschirm sind, da Sie darin den Bildausschnitt mit Rollbalken verschieben können. Für Text, Bilder, Kalkulationsblätter oder ähnliches sind sie prädestiniert.

Für jedes Fenster legen Sie nach einem Baukastenprinzip fest, welche Attribute es haben soll. Zur Auswahl stehen Rollbalken (Slider) zum Verschieben, eine Infozeile, veränderbare Größe, Verschiebbarkeit und anderes mehr. Die Konstanten für die gewünschten Attribute verknüpfen Sie mit »Oder« und übergeben sie mitsamt dem Fenstertitel und der Startgröße des Fensters an die Funktion »New_Window«. Als Rückgabewert erhalten Sie das sogenannte Window-Handle, eine Ganzzahl, mit der Sie von nun an auf dieses Fenster zugreifen können. GEM sorgt dafür, daß stets alle Fenster unterschiedliche Handles haben.

Mit »Open_Window« stellen Sie das Fenster dar. Hierbei zeichnet GEM nur die gewählten Rahmenelemente, den Inhalt des Fensters muß danach Ihr Programm selbst erzeugen. Das Schließen des Fensters erfolgt mit »Close_Window«. Ebenso wie Dialo-

DMV-PROFITOOLS FÜR CLIPPER

Statt 100 simpler Funktionen ein professionelles Entwicklungssystem

Die vorliegenden Clipper-Tools entstanden aus der Erfahrung und der täglichen Arbeit mit professionellen Projekten. Sie stellen nun eine standardisierte Lösung der immer wiederkehrenden Aufgaben dar. Im Laufe der Zeit konnten die Routinen in vielen Bereichen optimiert werden, so daß mit den DMV-ProfiTools für Clipper eine ausgereifte Programmsammlung für "Clipper Sommer '87" und "Clipper 5" zur Verfügung steht. Wie die grafische Darstellung zeigt, decken die ProfiTools alle Problembereiche der Clipper-Programmentwicklung ab. Sie enthalten sämtliche Mechanismen komplexer Applikationen und bilden als Gesamtes ein umfassendes Entwicklungssystem.

Um alle Funktionen anzusprechen, reicht der Platz hier bei weitem nicht. Um Sie aber neugierig zu machen, seien einige der Leistungsmerkmale herausgestellt. Neben einem **Window-System, integrierter Netzwerkfähigkeit, einem DOS-Manager, automatischer Farbanpassung** und vielem, vielem anderen bieten die ProfiTools:

Toggles: Sie markieren und verwalten komfortabel Anfang und Ende von Datenbankausschnitten und ersetzen die umständlicheren, zeitraubenden Filter.

Standard-Verwaltung: Ein zentrales Modul, dem alle notwendigen Funktionen angegliedert sind, verkürzt den Entwicklungsaufwand der Datenbankentwicklung auf ein Minimum.

Übersicht: Toggling, Marking, Seitwärts-Scrolling, freie Programmierung und viele weitere Details gewähren präzise und komplette Überschaubarkeit der Datenbank mit diesen Tools.

Hilfesystem: Der intelligente Hilfegenerator erzeugt eine hierarchische Textgliederung, die interaktiv und in verschiedenen Applikationen aufgerufen werden kann.

Marking: Ein neues Verfahren der Datensatzmarkierung erhöht bei komplexen Selektionen enorm die Arbeitsgeschwindigkeit. Bis zu 500.000 Datensätze können pro Datei im RAM gehalten werden.

Druckmanager: Auch mehrfach geschachtelte Auswertungen kann der Drucklisten-Manager einfach und schnell bearbeiten.

Maskengenerator: derzeit das Optimum an Arbeitserleichterung.

DMV-ProfiTools für Clipper **199,- DM***

außerdem die Entwicklungsumgebung mit Editor:

DMV-Integrator für Clipper **199,- DM***

* Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland DM 4,- bzw. für das Ausland DM 6,- Porto und Verpackung.

Clipper ist ein eingetragenes Warenzeichen der Nantucket Corporation.

DMV-Verlag • Postfach 250 • 3440 Eschwege

DMV Daten- und Medienverlag

Code	Bedeutung
0	kein Symbol
1	Achtung
2	Frage
3	Stop

Tabelle 2. Diese Codes steuern, welches Symbol eine Alert-Box anzeigt

ge und Menüs sollten Sie auch Fenster bei GEM abmelden. Dies geschieht mit »Delete_Window«.

Genau wie Menüs verwaltet GEM die Fenster ereignisgesteuert. Hierzu rufen Sie »Get_Event« in einer Schleife immer wieder auf. Im Rückgabewert sind die den aufgetretenen Ereignissen entsprechenden Bits gesetzt. Sie müssen also zunächst feststellen, welches Ereignis aufgetreten ist. Tabelle 3 enthält die Alternativen, die es hier gibt. In »gembind.tpu« sind diese Werte als Konstanten bereits definiert; Sie können deshalb die symbolischen Namen verwenden. Das häufigste Ereignis beim Arbeiten mit Menüs und Fenstern ist die GEM-Meldung. In den ersten zwei Bytes des Meldungspuffers steht nun als 16-Bit-Zahl, um welche Meldung es sich handelt. Hiervon hängt auch die Bedeutung der restlichen Werte des Meldungspuffers ab. Die in Tabelle 4 aufgeführten Meldungen, die ebenfalls in der Unit vordefiniert sind, können auftreten.

Es bleibt dem Programm, also Ihnen, überlassen, auf welche Meldungen es reagiert und welche es nicht beachtet. Listing 3 zeigt ein typisches Beispiel für die Programmierung eines Fensters. Beachten Sie, wie sich die Prozedur »Draw_Window« immer mit »Work_Rect« die Koordinaten des zu zeichnenden Rechtecks holt und nie mit absoluten Werten arbeitet. Das ist wichtig, damit das Programm auf allen Grafikkarten korrekt läuft. Achten Sie unbedingt darauf, daß Sie die Maus mit »Init_Mouse« initialisieren, bevor Sie die Prozeduren »Hide_Mouse« oder »Show_Mouse« verwenden. Sonst ist ein Programmabsturz die unweigerliche Folge. Das Programm wartet auf verschiedene Ereignisse, die eine Veränderung der Größe des Fensters anzeigen. Dann baut das Programm das Fenster mit den neuen Koordinaten oder Größenangaben neu auf und zeichnet gegebenenfalls den Inhalt des Fensters, eine Raute, neu. Es beachtet ab jetzt die Meldungen »WM_Redraw«, »WM_Sized«, »WM_Moved« und »WM_Fulled«. Auf die Meldung »WM_Closed« schließt es das Fenster und beendet sich. Sie können das Fenster mit der Maus vergrößern, verkleinern und verschieben (Bilder 2 und 3). Bei manchen Versionen von GEM bleibt an der Startposition der Maus ein »Schatten« zurück, wenn Sie die Maus bewegen. Schieben Sie einmal das Fenster über diesen Bereich, dann ist der Schatten verschwunden. Der Grund für diese Anomalie ist unbekannt.

Um Grafikausgaben mit dem VDI machen zu können, ist es notwendig, auf einem physikalischen Gerät eine virtuelle Workstation zu eröffnen. Mit dieser Methode können sich mehrere gleichzeitig laufende Programme ein physikalisches Gerät teilen. Da die Verwendung von virtuellen Workstations aber recht umständlich sein kann, eröffnet »Init_Gem« automatisch eine Workstation und verwaltet sie. Sie können nun die Grafikbefehle »Paint_Rect«, »Pline« etc. benutzen, als ob Sie direkt auf dem physikalischen Bildschirm arbeiten würden. Falls Sie einmal mit mehreren oder anderen Workstations arbeiten möchten, können Sie diese mit dem VDI-Aufruf 100 selbst öffnen und dann die Handle in der Variablen »VDI_Handle« ablegen.

Auch bei komplexen Problemstellungen läßt GEM Sie nicht im Stich. Die Auswahl einer Datei aus einer Liste der auf dem Massenspeicher vorhandenen Dateien mit einer bestimmten Dateierweiterung innerhalb einer Dialogbox ist nicht aufwendig (Listing 4). Ein einziger GEM-Aufruf reicht aus. Sie übergeben ihm den gewünschten Pfad mit den unter MS-DOS üblichen Stellvertreterzeichen (»*« und »?«) (Bild 4).

Die MS-DOS-Version von GEM unterscheidet sich im Detail von der von Atari-Computern

Die meiste Literatur zum GEM gibt es zweifellos für den Atari ST, da dieser Rechner GEM fest eingebaut hat. Es gibt nun ein paar Klippen, auf die Sie beim Übertragen des eigenen GEM-Wissens vom ST auf den PC stoßen werden. Aufgrund der sehr unterschiedlichen Hardwarekonfigurationen, mit denen das PC-GEM zurechtkommen muß, stellt es härtere Anforderungen an eine saubere Programmierung. Ein Programm muß sich flexibel an verschiedene Auflösungen anpassen. Hierzu stellt Ihnen »gembind.tpu« die Variablen »GEMResX« und »GEMResY« zur Verfügung. Diese füllt »Init_Gem« mit den Werten der aktuellen Bildschirmauflösung. Beim Anpassen von Programmen, die auf dem Atari in hoher Auflösung (640x400 Pixel) gelaufen sind, haben sich folgende Formeln bewährt:

```
Neu_X:= Round(Alt_X * (GEMResX / 639));
```

und

```
Neu_Y := Round(Alt_Y * (GEMResY / 399));
```

Wenn Sie es vom Atari ST gewöhnt sind, Textein- und -ausgaben in GEM-Programmen mit »ReadLn« und »WriteLn« zu erledigen, werden Sie dies auf dem PC vermissen. Der Atari ist hier wesentlich toleranter, da bei ihm Text- und Grafikbildschirm identisch sind. Auf dem PC müssen Sie alle Ein- und Ausgaben über GEM-Funktionen erledigen. Anders als das Atari-GEM scheint das PC-GEM den für Dateiauswahlboxen belegten Speicher nicht sofort wieder freizugeben, nachdem ein Programm diese geschlossen hat. Seien Sie also vorsichtig bei häufigen Aufrufen von »Get_In_File« bei knappem Speicher.

Ein Grundproblem, das Sie beim Übertragen von Programmen vom Atari auf den PC im Auge behalten

Name	Bedeutung
E_Keyboard	eine Taste wurde gedrückt
E_Button	ein Mausknopf wurde gedrückt
E_Mouse1	der Mauszeiger hat Bereich 1 verlassen
E_Mouse2	der Mauszeiger hat Bereich 2 verlassen
E_Message	eine Message von GEM ist vorhanden
E_Timer	der eingestellte Timer ist abgelaufen

Tabelle 3. Diese Ereignisse (Events) können unter GEM auftreten. Das Ereignis »E_Message« zeigt an, daß eine der Meldungen aus der Tabelle 3 eingetroffen ist.

Name	Bedeutung
MN_selected	ein Menüeintrag ist gewählt worden
WM_redraw	ein Teil eines Fensters muß neu gezeichnet werden
WM_topped	das Fenster ist zum aktuellen geworden
WM_closed	ein Fenster soll geschlossen werden
WM_fulled	ein Fenster soll auf volle Größe gebracht werden
WM_arrow	in einem Fenster soll gerollt werden
WM_hslid	ein horizontaler Rollbalken wurde bewegt
WM_vslid	ein vertikaler Rollbalken wurde bewegt
WM_sized	ein Fenster soll in der Größe verändert werden
WM_moved	ein Fenster soll bewegt werden
WM_newtop	ein Fenster wurde aktiviert
WM_untopped	ein Fenster wird von einem anderen überlagert
AC_open	ein Desk-Accessoire ist geöffnet worden
AC_close	ein Desk-Accessoire ist geschlossen worden

Tabelle 4. GEM informiert Ihr Programm mit Meldungen über die Aktionen des Benutzers

sollten, sind die unterschiedlichen Datenformate, mit denen die Prozessoren arbeiten. MS-DOS-Rechner legen die Daten im Intel-Format ab, während der ST das Motorola-Format verwendet. Beim PC sind die High- und Lowbytes in Worten vertauscht. Gleiches gilt für Worte in Langworten. Wichtig wird dies insbesondere beim Austauschen von ansonsten kompatiblen GEM-Dateien wie zum Beispiel Fonts und Ressourcedateien. Ein weiteres Problem beim Portieren ist die Speichersegmentierung auf MS-DOS-Rechnern. Turbo Pascal läßt im günstigsten Fall Variablen und Felder mit 64 KByte zu. Falls das zu wenig ist, müssen Sie die Daten auf zwei Variablen verteilen, die Sie in unterschiedlichen Segmenten oder auf dem Heap ablegen. Auch der Stack ist bei Turbo Pascal auf maximal 64 KByte begrenzt. Falls hier Probleme auftreten, müssen Sie einige lokale Variablen entweder global deklarieren oder dynamische Variablen benutzen, auf die Sie mit Zeigern zugreifen. Ein zweites oder ein größeres Stapelsegment können Sie nicht anlegen. Sie müssen also gerade bei stark verschachtelten Prozeduraufrufen darauf achten, daß der Stapel nicht zu groß wird.

Zum Anfertigen von Dialogen, Menüs und so weiter können Sie entweder das Resource Construction Set (RCS) unter MS-DOS verwenden (es ist Bestandteil des GEM Programmers Toolkit), oder aber Sie generieren die Ressourcen auf dem Atari ST und konvertieren sie dann. Das RCS für den Atari ist von Maxon Computer zu beziehen

Die Konvertierung können Sie mit dem Public-Domain-Programm »rsconv.prg« von Tim Oren auf dem Atari durchführen.

Die vorgestellte Bibliothek sollte für die meisten GEM-Anwendungen ausreichen. Falls Sie doch einmal eine nicht implementierte GEM-Funktion benötigen, können Sie diese via »AES_Call« und »VDI_Call« ohne größere Probleme selbst schreiben.

(Jan Willamowius/wn)

Literatur:
Jankowski, Reschke, Rabich: ATARI ST Profibuch, Sybex Verlag, 1991, 1300 Seiten, ISBN 3-88745-888-5, 79 Mark
Tim Oren: Professional GEM, Antic Publishing, 1986, Column 1-17
Info: Maxon Computer GmbH, 6236 Eschborn
Digital Research GmbH, 8000 München 81

```
[8V3]    1: (* Programm: gembind.pas
[ØL6]    2:    Funktion: Programmierschnittstelle zu
[BP2]    3:              GEM für Pascal
[2K4]    4:    Sprache:  Turbo-Pascal ab 5.Ø
[B24]    5:    Autor:    Jan Willamowius
[HA5]    6:    (C) 1991 DMV Widuch GmbH & Co KG      *)
[ØØØ]    7:
[L62]    8: unit Gembind;
[ØØØ]    9:
[SK1]   1Ø: interface
[ØØØ]   11:
[VG1]   12: uses Dos;
[ØØØ]   13:
[6XØ]   14: const
[PZ6]   15:   No_Window = -1;  {kein Fenster verfügbar)
[976]   16:   Root = Ø;        { Wurzel des Objektbaum)
[BU6]   17:   Null_Index = -1; { kein Objekt gefunden }
[WZ5]   18:   Max_Depth = 8;   { Maximale Schachte-
[T32]   19:                      lungstiefe }
[PØ7]   2Ø:   Max_Tree = 1ØØ;  { Maximale BaumgröLe }
[ØØØ]   21:
[KZ6]   22:   Max_C_String = 255; {Maximale Länge eines
[9G3]   23:                        C_String in Pascal }
[ØØØ]   24:
[TD6]   25:   Max_Fn = 12;     { Länge des Dateinamen }
[BC6]   26:   Max_Path = 8Ø;   { Länge des Pfadnamen }
[WM4]   27:   Max_WTitle = 8Ø; { Länge des
[1Z2]   28:                      Fenstertitels }
[ØØØ]   29:
[1L6]   3Ø:   R_Tree = Ø;      { Resourcestrukturen }
[142]   31:   R_String = 5;
[ØØØ]   32:
[TE2]   33:   E_Keyboard =  1;
[K92]   34:   E_Button   =  2;
[U82]   35:   E_Mouse1   =  4;
[B92]   36:   E_Mouse2   =  8;
[RC2]   37:   E_Message  = 16;
[AA2]   38:   E_Timer    = 32;
[ØØØ]   39:
[NX2]   4Ø:   Replace_Mode   = 1;
[CU2]   41:   Trans_Mode     = 2;
[2M2]   42:   XOR_Mode       = 3;
[W93]   43:   Rev_Trans_Mode = 4;
[ØØØ]   44:
[GX1]   45:   Solid      = 1;
[282]   46:   LongDash   = 2;
[H12]   47:   Dotted     = 3;
[972]   48:   DashDot    = 4;
[XZ1]   49:   Dashed     = 5;
[DG2]   5Ø:   DashDotDot = 6;
[ØØØ]   51:
[F72]   52:   Normal      =   Ø;
[YE2]   53:   Thickened   =   1;
[2F2]   54:   Lightened   =   2;
[PA2]   55:   Slanted     =   4;
[ML2]   56:   Underlined  =   8;
[NF2]   57:   Outlined    =  16;
[3D2]   58:   Shadowed    =  32;
[M42]   59:   Draw3D      =  64;
[EE2]   6Ø:   WhiteBak    = 128;
[ØØØ]   61:
[XU2]   62:   System_Font = 3;
[UG2]   63:   Small_Font  = 5;
[ØØØ]   64:
[U12]   65:   G_Box       = 2Ø;
[N82]   66:   G_Text      = 21;
[CG2]   67:   G_BoxText   = 22;
[K92]   68:   G_Image     = 23;
[XE2]   69:   G_ProgDef   = 24;
```

```
[F95]   7Ø:   { 2 Namen für die gleiche Sache }
[ZF2]   71:   G_UserDef   = 24;
[472]   72:   G_IBox      = 25;
[4E2]   73:   G_Button    = 26;
[FE2]   74:   G_Boxchar   = 27;
[TE2]   75:   G_String    = 28;
[AD2]   76:   G_FText     = 29;
[SP2]   77:   G_FBoxText  = 3Ø;
[762]   78:   G_Icon      = 31;
[KC2]   79:   G_Title     = 32;
[ØØØ]   8Ø:
[W72]   81:   None        = $ØØØ;
[KT2]   82:   Selectable  = $ØØ1;
[AL2]   83:   Default     = $ØØ2;
[ER2]   84:   Exit_Btn    = $ØØ4;
[TK2]   85:   Editable    = $ØØ8;
[BR2]   86:   Radio_Btn   = $Ø1Ø;
[8M2]   87:   Last_Ob     = $Ø2Ø;
[HZ2]   88:   Touch_Exit  = $Ø4Ø;
[RS2]   89:   Hide_Tree   = $Ø8Ø;
[ZK2]   9Ø:   Indirect    = $1ØØ;
[ØØØ]   91:
[X62]   92:   Selected    = 1;
[D62]   93:   Crossed     = 2;
[F32]   94:   Checked     = 4;
[572]   95:   Disabled    = 8;
[ØØØ]   96:
[6F2]   97:   G_Name      = $ØØ1;
[AM2]   98:   G_Close     = $ØØ2;
[PF2]   99:   G_Full      = $ØØ4;
[RG2]  1ØØ:   G_Move      = $ØØ8;
[EF2]  1Ø1:   G_Info      = $Ø1Ø;
[XG2]  1Ø2:   G_Size      = $Ø2Ø;
[AW2]  1Ø3:   G_UpArrow   = $Ø4Ø;
[CU2]  1Ø4:   G_DnArrow   = $Ø8Ø;
[NP2]  1Ø5:   G_VSlide    = $1ØØ;
[1S2]  1Ø6:   G_LArrow    = $2ØØ;
[VS2]  1Ø7:   G_RArrow    = $4ØØ;
[4N2]  1Ø8:   G_HSlide    = $8ØØ;
[BE2]  1Ø9:   G_All       = $FEF;
[ØØØ]  11Ø:
[8M1]  111:   White   =  Ø;
[6K1]  112:   Black   =  1;
[5F1]  113:   Red     =  2;
[WL1]  114:   Green   =  3;
[5K1]  115:   Blue    =  4;
[8H1]  116:   Cyan    =  5;
[6U1]  117:   Yellow  =  6;
[1W1]  118:   Magenta =  7;
[312]  119:   L_White =  8;
[9X1]  12Ø:   L_Black =  9;
[ES1]  121:   L_Red   = 1Ø;
[TØ2]  122:   L_Green = 11;
[8V1]  123:   L_Blue  = 12;
[DV1]  124:   L_Cyan  = 13;
[E92]  125:   L_Yellow = 14;
[PF2]  126:   L_Magenta = 15;
[ØØØ]  127:
[3S2]  128:   MN_Selected = 1Ø;
[SH2]  129:   WM_Redraw   = 2Ø;
[9K2]  13Ø:   WM_Topped   = 21;
[WH2]  131:   WM_Closed   = 22;
[DH2]  132:   WM_Fulled   = 23;
[1P2]  133:   WM_Arrowed  = 24;
[SE2]  134:   WM_HSlid    = 25;
[7F2]  135:   WM_VSlid    = 26;
```

Listing 1. Die Unit »gembind.pas« ist die Schnittstelle zwischen Turbo Pascal und GEM

```
[BF2]  136:    WM_Sized    = 27;
[6F2]  137:    WM_Moved    = 28;
[YL2]  138:    WM_Newtop   = 29;
[NV2]  139:    WM_Untopped = 3Ø;
[DB2]  14Ø:    AC_Open     = 4Ø;
[HD2]  141:    AC_Close    = 41;
[ØØØ]  142:
[M92]  143:    WF_Kind      =  1;
[992]  144:    WF_Name      =  2;
[KA2]  145:    WF_Info      =  3;
[6V2]  146:    WF_WorkXYWH  =  4;
[VV2]  147:    WF_CurrXYWH  =  5;
[CV2]  148:    WF_PrevXYWH  =  6;
[PU2]  149:    WF_FullXYWH  =  7;
[8E2]  15Ø:    WF_HSlide    =  8;
[RE2]  151:    WF_VSlide    =  9;
[282]  152:    WF_Top       = 1Ø;
[443]  153:    WF_FirstXYWH = 11;
[1Y2]  154:    WF_NextXYWH  = 12;
[XM2]  155:    WF_Newdesk   = 14;
[FP2]  156:    WF_HSlSize   = 15;
[YP2]  157:    WF_VSlSize   = 16;
[BG2]  158:    WF_Screen    = 17;
[ØK2]  159:    WF_TAttrb    = 18;
[VL2]  16Ø:    WF_SizTop    = 19;
[ØØØ]  161:
[GU4]  162:    Crystal: array [1Ø..125,1..3] of
[GU1]  163:      Integer = (
[DT3]  164:    (Ø,1,Ø), (2,1,1), (2,1,1), (Ø,1,1),
[6T3]  165:    (2,1,1), (1,1,1), (Ø,Ø,Ø), (Ø,Ø,Ø),
[HT3]  166:    (Ø,Ø,Ø), (Ø,1,Ø), (Ø,1,Ø), (3,5,Ø),
[WV3]  167:    (5,5,Ø), (Ø,1,1), (2,1,Ø), (16,7,1),
[TS3]  168:    (2,1,Ø), (Ø,Ø,Ø), (Ø,Ø,Ø), (Ø,Ø,Ø),
[RT3]  169:    (1,1,1), (2,1,1), (2,1,1), (2,1,1),
[6T3]  17Ø:    (1,1,2), (1,1,1), (Ø,Ø,Ø), (Ø,Ø,Ø),
[ØT3]  171:    (Ø,Ø,Ø), (Ø,Ø,Ø), (2,1,1), (1,1,1),
[LU3]  172:    (6,1,1), (4,1,1), (1,3,1), (2,1,1),
[9U3]  173:    (4,2,1), (8,1,1), (Ø,Ø,Ø), (Ø,Ø,Ø),
[AU3]  174:    (1,1,1), (9,1,Ø), (1,1,1), (1,1,Ø),
[TT3]  175:    (Ø,5,1), (3,3,1), (2,2,1), (Ø,Ø,Ø),
[KS3]  176:    (Ø,Ø,Ø), (Ø,Ø,Ø), (Ø,Ø,Ø), (Ø,Ø,Ø),
[KS3]  177:    (Ø,Ø,Ø), (Ø,Ø,Ø), (Ø,Ø,Ø), (Ø,Ø,Ø),
[KS3]  178:    (Ø,Ø,Ø), (Ø,Ø,Ø), (Ø,Ø,Ø), (Ø,Ø,Ø),
[4V3]  179:    (4,3,Ø), (8,3,Ø), (6,1,Ø), (8,1,Ø),
[WU3]  18Ø:    (8,1,Ø), (4,1,1), (3,1,1), (Ø,5,Ø),
[DT3]  181:    (1,1,1), (Ø,5,Ø), (Ø,1,1), (Ø,1,1),
[KS3]  182:    (Ø,Ø,Ø), (Ø,Ø,Ø), (Ø,Ø,Ø), (Ø,Ø,Ø),
[KS3]  183:    (Ø,Ø,Ø), (Ø,Ø,Ø), (Ø,Ø,Ø), (Ø,Ø,Ø),
[US3]  184:    (Ø,2,2), (Ø,Ø,Ø), (Ø,Ø,Ø), (Ø,Ø,Ø),
[KS3]  185:    (Ø,Ø,Ø), (Ø,Ø,Ø), (Ø,Ø,Ø), (Ø,Ø,Ø),
[8T3]  186:    (Ø,Ø,Ø), (Ø,Ø,Ø), (5,1,Ø), (5,1,Ø),
[XT3]  187:    (1,1,Ø), (1,1,Ø), (2,5,Ø), (6,1,Ø),
[MT3]  188:    (2,1,Ø), (1,1,Ø), (6,5,Ø), (Ø,Ø,Ø),
[7T3]  189:    (Ø,1,1), (Ø,1,Ø), (2,1,Ø), (2,1,1),
[TS3]  19Ø:    (1,1,1), (Ø,Ø,Ø), (Ø,Ø,Ø), (Ø,Ø,Ø),
[5T3]  191:    (Ø,Ø,Ø), (Ø,Ø,Ø), (Ø,1,2), (3,1,2),
[YU3]  192:    (1,1,1), (1,1,1), (Ø,1,1), (Ø,1,2));
[ØØØ]  193:
[SW5]  194: type Color_Reg = White..L_Magenta;
[Y12]  195:      Draw_Modes =
[Ø75]  196:        Replace_Mode..Rev_Trans_Mode;
[HG5]  197:      Line_Types = Solid..DashDotDot;
[RW3]  198:      Paint_Types = Integer;
[ØØØ]  199:
[B26]  2ØØ:      Mouse_Type = (M_Arrow,M_Text_Curs,
[2F3]  2Ø1:                    M_Bee,M_Point_Hand,
[582]  2Ø2:                    M_Flat_Hand,
[KG2]  2Ø3:                    M_Thin_Cross,
[AL2]  2Ø4:                    M_Thick_Cross,
[3W2]  2Ø5:                    M_Outln_Cross);
[ØØØ]  2Ø6:
[943]  2Ø7:      Pointer     = ^Char;
[693]  2Ø8:      Text_Ptr    = ^Char;
[ØØØ]  2Ø9:
[172]  21Ø:      Mouse_Form  =
[Y11]  211:      record
[KN2]  212:        hot_x: Integer;
[MN2]  213:        hot_y: Integer;
[V62]  214:        res: Integer;
[GM3]  215:        mask_color: Integer;
[SL3]  216:        data_color: Integer;
[235]  217:        mask: array [Ø..15] of Integer;
[A25]  218:        data: array [Ø..15] of Integer;
[ZNØ]  219:      end;
[ØØØ]  22Ø:
[VB5]  221:      File_Name         = string [Max_Fn];
[TV5]  222:      Path_Name         = string [Max_Path];
[4R6]  223:      Window_Title      = string [Max_WTitle];
[ØØØ]  224:
[F45]  225:      Message_Buffer    = array [Ø..15]
[SW1]  226:                          of Integer;
[ØØØ]  227:
[8P4]  228:      Ctrl_Parms        = array [Ø.. 11]
[SW1]  229:                          of Integer;
[CX4]  23Ø:      Global_parms      = array [Ø.. 15]
[SW1]  231:                          of Integer;
[E15]  232:      Int_In_Parms      = array [Ø..128]
[SW1]  233:                          of Integer;
[H45]  234:      Int_Out_Parms     = array [Ø..128]
[SW1]  235:                          of Integer;
```


Große Software zum erstaunlich kleinen Preis!

DMV Soft-Hits

MultiCopy

Nie mehr Diskjockey: Disketten duplizieren in einem Rutsch! Multicopy liest die Originaldiskette komplett in den Speicher. Dann können Sie beliebig viele Kopien auf die Zieldisketten ziehen. Multicopy puffert die Daten entweder im vorhandenen EMS-Speicher oder in einer Datei auf der Festplatte. Da man mehrere Pufferdateien benennen kann, lassen sich ganze Serien von Disketten speichern, die man auf Wunsch wieder auf die Disketten zurückschreiben kann: Zeitersparnis "hoch 3". Natürlich werden die Zieldisketten bei Bedarf auch gleich formatiert. Komfortable Bedienung der SAA-Oberfläche mit und ohne Maus.

DataBase

DataBase ist eine sehr leistungsfähige Datenbank mit allem, was man von einem Datenverwaltungsprogramm erwartet: Maskengestaltung, Eingeben, Suchen, Löschen, Selektieren, Drucken, Dateiausgabe und Hilfefunktion. Mit den umfangreichen Reportfunktionen werden die Datenausgaben ganz individuell gestaltet. Drei verschiedene Dateimasken sind schon dabei, eine Adreßverwaltung, eine Literaturverwaltung und eine Kundendatei. Der selbsterklärende Menüaufbau macht die Bedienung der Datenbank zum Kinderspiel.

Ram-Patrouille

Laden und Entfernen von residenten Programmen. Behalten Sie die Kontrolle über den RAM-Speicher Ihres PC! Mit RAM-Patrouille ist es kein Problem, solche Programme, die im RAM verbleiben, zu überwachen. RAM-Patrouille managed den Speicher: Laden der residenten Programme, Platzbedarf kontrollieren und der wichtigste Punkt, nämlich wieder aus dem Speicher entfernen, ohne daß der Rechner neu gebootet werden muß: RAM frei nach Bedarf! Zu bedienen ist das Programm sehr komfortabel über Menüs oder für die Kommandozeilen-Freunde am DOS-Prompt.

Tippsi

Maschinenschreiben leichtgemacht: Tippsi übt mit Ihnen das Zehnfingersystem. Dabei werden Sie nach professionellen Methoden trainiert, und zwar nach Lektionen, die auch individuell verändert werden können. Tippsi kontrolliert Ihre Fortschritte und protokolliert Fehler und Ausführungszeiten. "Vom Leichten zum Schweren" werden Sie durch die Übungen geführt, so daß perfektes Maschinenschreiben binnen kürzester Zeit erlernbar ist.

Demnächst in derselben Reihe:

"Menue": gestaltet die individuelle Benutzeroberfläche; "Astro": das "PC-Globe" für den Sternenhimmel; "TextCompiler": macht aus jedem Text ein ausführbares Programm.

Jeder Soft-Hit nur 49,– *

* Unabhängig von der Anzahl der bestellten Programme berechnen wir für das Inland DM 4,– bzw. für das Ausland DM 6,– Porto und Verpackung. Bitte benutzen Sie die Bestellkarte.

DMV-Verlag • Postfach 250 • 3440 Eschwege

DMV Daten- und Medienverlag

```
[D25]  236:      Pts_In_Parms    = array [Ø..256]
[SW1]  237:                        of Integer;
[C35]  238:      Pts_Out_Parms   = array [Ø.. 11]
[SW1]  239:                        of Integer;
[AW4]  24Ø:      Addr_In_Parms   = array [Ø..  2]
[HX1]  241:                        of Pointer;
[625]  242:      Addr_Out_Parms  = array [Ø..  2]
[HX1]  243:                        of Pointer;
[ØØØ]  244:
[TA2]  245:      Point = record
[Y12]  246:              x,y: Integer
[ZNØ]  247:             end;
[ØØØ]  248:
[3Z2]  249:      Rectangle = record
[KT2]  25Ø:                    x,y,w,h: Integer;
[ZNØ]  251:                  end;
[ØØØ]  252:
[SB4]  253:      Str255          = string [255];
[ØL5]  254:      TE_Just         = (TE_Left,TE_Right,
[A22]  255:                         TE_Center);
[BT4]  256:      Tree_Range = Root..Max_Tree;
[K26]  257:      Tree_Index = Null_Index..Max_Tree;
[ØØØ]  258:
[S25]  259:      Ob_Type         = G_Box..G_Title;
[ØØØ]  26Ø:
[DY3]  261:      C_String = Packed array
[4F4]  262:                 [Ø..Max_C_String] of Char;
[FN4]  263:      String_Ptr      = ^C_String;
[ØØØ]  264:
[1G2]  265:      Text_Ed_Info =
[Y11]  266:      record
[6X3]  267:        te_ptext: String_Ptr;
[B14]  268:        te_ptmplt: String_Ptr;
[SY3]  269:        te_pvalid: String_Ptr;
[D13]  27Ø:        te_font: Integer;
[153]  271:        te_junk1: Integer;
[C23]  272:        TE_Just: Integer;
[T53]  273:        te_color: Integer;
[253]  274:        te_junk2: Integer;
[6W3]  275:        te_thickness: Integer;
[UE3]  276:        te_txtlen: Integer;
[SD3]  277:        te_tmplen: Integer;
[ZNØ]  278:      end;
[HE4]  279:      Ted_Ptr = ^Text_Ed_Info;
[ØØØ]  28Ø:
[PZ1]  281:      Spec_Info =
[Y11]  282:      record
[BR2]  283:        case Ob_Type of
[P11]  284:          G_Box,
[7B1]  285:          G_IBox,
[9H4]  286:          G_Boxchar: (thick: Integer;
[ZK2]  287:          color: Integer);
[WC1]  288:          G_Text,
[6U1]  289:          G_BoxText,
[5H1]  29Ø:          G_FText,
[WU4]  291:          G_FBoxText: (info: Ted_Ptr);
[LD1]  292:          G_Image,
[NS1]  293:          G_ProgDef,
[EM1]  294:          G_Button,
[3Ø4]  295:          G_Icon: (Ptr: Pointer);
[1N1]  296:          G_String,
[NR4]  297:          G_Title: (Str: String_Ptr);
[ZNØ]  298:        end;
[ØØØ]  299:
[Ø91]  3ØØ:      Objekt =
[Y11]  3Ø1:      record
[C13]  3Ø2:        ob_next: Integer;
[1X2]  3Ø3:        ob_head: Integer;
[XY2]  3Ø4:        ob_tail: Integer;
[6Ø3]  3Ø5:        ob_Type: Integer;
[333]  3Ø6:        ob_flags: Integer;
[653]  3Ø7:        ob_state: Integer;
[ØH3]  3Ø8:        ob_spec: Spec_Info;
[BD2]  3Ø9:        ob_x: Integer;
[CD2]  31Ø:        ob_y: Integer;
[AD2]  311:        ob_w: Integer;
[VC2]  312:        ob_h: Integer;
[ZNØ]  313:      end;
[ØØØ]  314:
[395]  315:      Tree            = array [Tree_Range]
[TK1]  316:                        of Objekt;
[FH3]  317:      Tree_Ptr        = ^Tree;
[H74]  318:      Menu_Ptr        = Tree_Ptr;
[VC4]  319:      Dialog_Ptr      = Tree_Ptr;
[ØØØ]  32Ø:
[CØ2]  321:      Icon_Block =
[Y11]  322:      record
[173]  323:        ib_pmask: Pointer;
[963]  324:        ib_pdata: Pointer;
[S83]  325:        ib_ptext: Pointer;
[EY2]  326:        ib_char: Integer;
[L23]  327:        ib_xchar: Integer;
[M23]  328:        ib_ychar: Integer;
[443]  329:        ib_xicon: Integer;
[543]  33Ø:        ib_yicon: Integer;
[343]  331:        ib_wicon: Integer;
[M33]  332:        ib_hicon: Integer;
[A63]  333:        ib_xtext: Integer;
[B63]  334:        ib_ytext: Integer;
[963]  335:        ib_wtext: Integer;
[U53]  336:        ib_htext: Integer;
[ZNØ]  337:      end;
[ØØØ]  338:
[ZY1]  339:      Bit_Block =
[Y11]  34Ø:      record
[G63]  341:        bi_pdata: Pointer;
[CM2]  342:        bi_wb: Integer;
[HM2]  343:        bi_hl: Integer;
[6D2]  344:        bi_x: Integer;
[7D2]  345:        bi_y: Integer;
[153]  346:        bi_color: Integer;
[ZNØ]  347:      end;
[ØØØ]  348:
[ER1]  349:      Parm_Blk =
[Y11]  35Ø:      record
[Ø13]  351:        pb_tree: Pointer;
[CW2]  352:        pb_obj: Integer;
[3Y3]  353:        pb_prevstate: Integer;
[LY3]  354:        pb_currstate: Integer;
[DD2]  355:        pb_x: Integer;
[ED2]  356:        pb_y: Integer;
[CD2]  357:        pb_w: Integer;
[XC2]  358:        pb_h: Integer;
[PM2]  359:        pb_xc: Integer;
[RM2]  36Ø:        pb_yc: Integer;
[NM2]  361:        pb_wc: Integer;
[6M2]  362:        pb_hc: Integer;
[123]  363:        pb_parm: Pointer;
[ZNØ]  364:      end;
[ØØØ]  365:
[4E4]  366: var Control: Ctrl_Parms;
[NR3]  367:     Global: Global_parms;
[RW3]  368:     Int_In: Int_In_Parms;
[FE4]  369:     Int_Out: Int_Out_Parms;
[864]  37Ø:     Addr_In: Addr_In_Parms;
[FK4]  371:     Addr_Out: Addr_Out_Parms;
[FY3]  372:     Pts_In: Pts_In_Parms;
[RG4]  373:     Pts_Out: Pts_Out_Parms;
[1Z1]  374:     VDI_Handle,
[KF1]  375:     GEMResX,
[6Z2]  376:     GEMResY: Integer;
[ØØØ]  377:
[ØØØ]  378:
[ZF5]  379: procedure AES_Call (op: Integer;
[EL4]  38Ø:                   var Int_In: Int_In_Parms;
[655]  381:                   var Int_Out: Int_Out_Parms;
[ZW4]  382:                   var Addr_In: Addr_In_Parms;
[LD5]  383:                   var Addr_Out: Addr_Out_Parms);
[ØØØ]  384:
[5Ø7]  385: procedure VDI_call (cmd,sub_cmd: Integer;
[FK3]  386:                   nints,npts: Integer;
[4E4]  387:                   var Control: Ctrl_Parms;
[EL4]  388:                   var Int_In: Int_In_Parms;
[655]  389:                   var Int_Out: Int_Out_Parms;
[6N4]  39Ø:                   var Pts_In: Pts_In_Parms;
[E75]  391:                   var Pts_Out: Pts_Out_Parms;
[R83]  392:                   translate: Boolean);
[ØØØ]  393:
[ØZ4]  394: function  Init_GEM: Integer;
[NG3]  395: procedure Exit_GEM;
[8T3]  396: procedure Hide_Mouse;
[ZW3]  397: procedure Show_Mouse;
[MF7]  398: procedure Set_Mouse (M_Type: Mouse_Type);
[8V3]  399: procedure Init_Mouse;
[5G5]  4ØØ: function  Do_Alert (maske: Str255;
[6W3]  4Ø1:                     btn: Integer): Integer;
[R25]  4Ø2: function  Load_Resource(name:
[2U2]  4Ø3:                     string): Boolean;
[WA4]  4Ø4: procedure Free_Resource;
[7D6]  4Ø5: procedure Find_Menu (Index: Integer;
[YF3]  4Ø6:                     var Menu: Menu_Ptr);
[3D6]  4Ø7: procedure Find_Dialog(Index: Integer;
[LR3]  4Ø8:                     var Dial: Dialog_Ptr);
[537]  4Ø9: procedure Objc_Draw (dialog: Dialog_Ptr;
[SØ5]  41Ø:                     start,tiefe,x,y,w,h: Integer);
[EX6]  411: function  Do_dialog (dialog: Dialog_Ptr;
[2V4]  412:                     ed_start: Integer): Integer;
[AC7]  413: procedure Show_Dialog (dialog: Dialog_Ptr);
[3D7]  414: procedure Center_Dialog (Dial: Dialog_Ptr);
[757]  415: procedure End_dialog (dialog: Dialog_Ptr);
[AN6]  416: procedure Set_DText (Dial: Dialog_Ptr;
[ØN4]  417:                     item: Tree_Index; s: Str255;
[M25]  418:                     font: Integer; just: TE_Just);
[V47]  419: procedure Obj_SetState (Dial: Dialog_Ptr;
[VØ3]  42Ø:                     Index: Tree_Index;
[UK2]  421:                     state: Integer;
[1W2]  422:                     redraw: Boolean);
[7H6]  423: function  Obj_State (Dial: Dialog_Ptr;
[B14]  424:                     Index: Integer): Integer;
[WH6]  425: procedure Get_DEdit (Dial: Dialog_Ptr;
[1D5]  426:                     item: Tree_Index; var s: Str255);
[XF6]  427: procedure Draw_Menu (Menu: Menu_Ptr);
[2L6]  428: procedure Erase_Menu (Menu: Menu_Ptr);
[NM6]  429: procedure Menu_Enable (Menu: Menu_Ptr;
[U23]  43Ø:                     item: Tree_Index);
[3Z6]  431: procedure Menu_Disable (Menu: Menu_Ptr;
[U23]  432:                     item: Tree_Index);
[2S6]  433: procedure Menu_Normal (Menu: Menu_Ptr;
```

Listing 1. Die Unit »gembind.pas« ist die Schnittstelle zwischen Turbo Pascal und GEM

```
[T73]  434:            Title: Tree_Index);
[WØ7]  435: procedure Menu_Hilight (Menu: Menu_Ptr;
[T73]  436:            Title: Tree_Index);
[TN6]  437: function  New_Window (w_type: Integer;
[Z84]  438:            var Title: Window_Title;
[PC3]  439:            x_max,y_max,w_max,
[274]  44Ø:            h_max:Integer): Integer;
[7P6]  441: procedure Open_window (Handle: Integer;
[ZX2]  442:            x,y,w,h: Integer);
[ED6]  443: procedure Set_WName (Handle: Integer;
[V44]  444:            var name: Window_Title);
[PE6]  445: procedure Set_Winfo (Handle: Integer;
[654]  446:            var info: Window_Title);
[Ø27]  447: procedure Close_Window (Handle: Integer);
[N87]  448: procedure Delete_Window (Handle: Integer);
[KE6]  449: procedure Work_Rect (Handle: Integer;
[PM3]  45Ø:            var x,y,w,h: Integer);
[NR5]  451: function  Front_Window: Integer;
[GS6]  452: procedure Set_Window (Handle: Integer);
[TL7]  453: procedure Wind_Get (Handle,request:Integer;
[234]  454:            var v1,v2,v3,v4: Integer);
[5M7]  455: procedure Wind_Set (Handle,request:Integer;
[BC3]  456:            v1,v2,v3,v4: Integer);
[X95]  457: function  Get_in_file (var Path,
[CB4]  458:            name: Path_Name): Boolean;
[N66]  459: procedure Dragbox (x1,y1,w1,h1,x2,y2,w2,
[PL3]  46Ø:            h2: Integer; var Endx,
[6G2]  461:            Endy: Integer);
[ZH6]  462: procedure Growbox (stx,sty,stw,sth,fix,
[PS3]  463:            fiy,fiw,fih: Integer);
[CA7]  464: procedure Shrinkbox (stx,sty,stw,sth,fix,
[PS3]  465:            fiy,fiw,fih: Integer);
[4X6]  466: function  Get_Event (emask,bmask,bstate,
[Y23]  467:            n_clicks: Integer;
[BL2]  468:            ticks: Longint;
[ØV2]  469:            m1_flag: Boolean;
[M44]  47Ø:            m1x,m1y,m1w,m1h: Integer;
[3V2]  471:            m2_flag: Boolean;
[Ø54]  472:            m2x,m2y,m2w,m2h: Integer;
[SY4]  473:            var message: Message_Buffer;
[2F4]  474:            var key,brtn,bclick,mx,my,
[HF4]  475:            kstate: Integer): Integer;
[KC6]  476: procedure Set_Clip (x,y,w,h: Integer);
[X66]  477: procedure Draw_String(x,y: Integer;
[LM1]  478:            s: Str255);
[HL7]  479: procedure Line_Color (color_index:Integer);
[N26]  48Ø: procedure Draw_Mode (mode: Integer);
[XT3]  481: procedure Paint_Color
[RZ3]  482:            (color_index: Integer);
[3W3]  483: procedure Paint_Style
[VØ4]  484:            (style_index: Integer);
[NP6]  485: procedure Paint_Rect (x,y,w,h: Integer);
[UA6]  486: procedure Line_Endstyle (beg_style,
[8C3]  487:            End_style: Integer);
[KA7]  488: procedure Line_Style (style: Line_Types);
[LB6]  489: procedure Pline (x1,y1,x2,y2: Integer);
[FL6]  49Ø: procedure Frame_Rect (x,y,w,h: Integer);
[Y37]  491: procedure First_Rect(hand: integer; var x,
[EM2]  492:            y, w, h: Integer);
[XV6]  493: procedure Next_Rect(hand: Integer; var x,
[EM2]  494:            y, w, h: Integer);
[DY5]  495: function Rect_Intersect(x1, y1, w1,
[8B4]  496:            h1: Integer; var x2, y2, w2,
[WK3]  497:            h2: Integer): Boolean;
[WT5]  498: procedure Set_WSize(hand, x, y, w,
[TU1]  499:            h: Integer);
[P44]  5ØØ: procedure Begin_Update;
[WR3]  5Ø1: procedure End_Update;
[4G6]  5Ø2: procedure Border_Rect(Handle: Integer;
[TW3]  5Ø3:            Var x, y, w, h: Integer);
[VP6]  5Ø4: procedure Text_Heigth (Heigth: Integer);
[YP6]  5Ø5: procedure Text_Style (style: Integer);
[997]  5Ø6: procedure Text_Rotation (angle: Integer);
[ØØØ]  5Ø7:
[ØØØ]  5Ø8:
[4L2]  5Ø9: Implementation
[ØØØ]  51Ø:
[AZ4]  511: procedure Pas_To_C (s: Str255;
[ZN3]  512:            var C_str: C_String);
[ZH2]  513: var i: Integer;
[MUØ]  514: begin
[XW3]  515:   for i := 1 to Length(s) do
[KC3]  516:   C_str[i-1] := s[i];
[RK4]  517:   C_str[Length(s)] := Chr(Ø);
[ZNØ]  518: end;
[ØØØ]  519:
[EB6]  52Ø: procedure C_to_pas (C_str: C_String;
[AB2]  521:            var s: Str255);
[ZH2]  522: var i: Integer;
[MUØ]  523: begin
[LYØ]  524:   i := Ø;
[RXØ]  525:   s := '';
[KB4]  526:   while C_str[i] <> Chr(Ø) do
[MUØ]  527:   begin
[1Z3]  528:     s := Concat(s,C_str[i]);
[WC1]  529:     i := i + 1;
[ZNØ]  53Ø:   end;
[ZNØ]  531: end;
[ØØØ]  532:
[Y64]  533: function dptr_to_point
[TR4]  534:           (dptr: Dialog_Ptr): Pointer;
```

```
[ØT1]  535: var Trick:
[Y11]  536: record
[BM2]  537:   case Integer of
[EY2]  538:     Ø: (d: Dialog_Ptr);
[U92]  539:     1: (p: Pointer);
[ZNØ]  54Ø:   end;
[MUØ]  541:   begin
[3L2]  542:     Trick.d := dptr;
[DA4]  543:     dptr_to_point := Trick.p;
[ZNØ]  544:   end;
[ØØØ]  545:
[T74]  546: function Mptr_to_point
[7F4]  547:          (Mptr: Menu_Ptr): Pointer;
[ØT1]  548: var Trick:
[Y11]  549: record
[BM2]  55Ø:   case Integer of
[NL2]  551:     Ø: (M: Menu_Ptr);
[U92]  552:     1: (p: Pointer);
[ZNØ]  553:   end;
[MUØ]  554:   begin
[GM2]  555:     Trick.M := Mptr;
[ZA4]  556:     Mptr_to_point := Trick.p;
[ZNØ]  557:   end;
[ØØØ]  558:
[876]  559: procedure split_addr (Addr: Pointer;
[R13]  56Ø:           var p1,p2: Integer);
[ØT1]  561: var Trick:
[Y11]  562: record
[BM2]  563:   case Integer of
[S92]  564:     Ø: (p: Pointer);
[6E2]  565:     1: (a1: Integer;
[N32]  566:     a2: Integer);
[ZNØ]  567:   end;
[MUØ]  568:   begin
[8K2]  569:     Trick.p := Addr;
[Z82]  57Ø:     p1 := Trick.a1;
[592]  571:     p2 := Trick.a2;
[ZNØ]  572:   end;
[ØØØ]  573:
[ZF5]  574: procedure AES_Call (op: Integer;
[EL4]  575:           var Int_In: Int_In_Parms;
[655]  576:           var Int_Out: Int_Out_Parms;
[ZW4]  577:           var Addr_In: Addr_In_Parms;
[LD5]  578:           var Addr_Out: Addr_Out_Parms);
[RNØ]  579: var
[LU2]  58Ø:   Regs: Registers;
[2Ø2]  581:   Addr_Block:
[Y11]  582:     record
[N54]  583:       Control_point: Pointer;
[GZ3]  584:       global_point: Pointer;
[124]  585:       Int_In_point: Pointer;
[D84]  586:       Int_Out_point: Pointer;
[P34]  587:       Addr_In_point: Pointer;
[FE4]  588:       Addr_Out_point: Pointer;
[ZNØ]  589:     end;
[MUØ]  59Ø: begin
[8T2]  591:   Control[Ø] := op;
[HW4]  592:   Control[1] := Crystal[op,1];
[PW4]  593:   Control[2] := Crystal[op,2];
[VW4]  594:   Control[3] := Crystal[op,3];
[MV4]  595:   Addr_Block.Control_point :=
[WB2]  596:     Addr(Control);
[WR6]  597:   Addr_Block.global_point := Addr(Global);
[2Z6]  598:   Addr_Block.Int_In_point := Addr(Int_In);
[RY4]  599:   Addr_Block.Int_Out_point :=
[XD2]  6ØØ:     Addr(Int_Out);
[XT4]  6Ø1:   Addr_Block.Addr_In_point :=
[DA2]  6Ø2:     Addr(Addr_In);
[6W4]  6Ø3:   Addr_Block.Addr_Out_point :=
[SL2]  6Ø4:     Addr(Addr_Out);
[RB4]  6Ø5:   Regs.ES := Seg (Addr_Block);
[4D4]  6Ø6:   Regs.BX := Ofs (Addr_Block);
[PU5]  6Ø7:   Regs.CX := 2ØØ;  { AES - Kennziffer }
[ED2]  6Ø8:   Intr ($EF,Regs);
[ZNØ]  6Ø9: end;
[ØØØ]  61Ø:
[5Ø7]  611: procedure VDI_call (cmd,sub_cmd: Integer;
[FK3]  612:           nints,npts: Integer;
[4E4]  613:           var Control: Ctrl_Parms;
[EL4]  614:           var Int_In: Int_In_Parms;
[655]  615:           var Int_Out: Int_Out_Parms;
[6N4]  616:           var Pts_In: Pts_In_Parms;
[E75]  617:           var Pts_Out: Pts_Out_Parms;
[R83]  618:           translate: Boolean);
[RNØ]  619: var
[LU2]  62Ø:   Regs: Registers;
[2Ø2]  621:   Addr_Block:
[Y11]  622:   record
[N54]  623:     Control_point: Pointer;
[124]  624:     Int_In_point: Pointer;
[X24]  625:     Pts_In_Point: Pointer;
[D84]  626:     Int_Out_point: Pointer;
[994]  627:     Pts_Out_point: Pointer;
[ZNØ]  628:   end;
[ØØØ]  629:
[MUØ]  63Ø: begin
[SX2]  631:   Control[Ø] := cmd;
[X93]  632:   Control[1] := npts;
[CE3]  633:   Control[3] := nints;
[4S3]  634:   Control[5] := sub_cmd;
```

```
[H94]  635:   Control[6] := VDI_Handle;
[MV4]  636:   Addr_Block.Control_point :=
[WB2]  637:     Addr(Control);
[547]  638:   Addr_Block.Int_In_point  := Addr(Int_In);
[167]  639:   Addr_Block.Pts_In_Point  := Addr(Pts_In);
[RY4]  64Ø:   Addr_Block.Int_Out_point :=
[XD2]  641:     Addr(Int_Out);
[7Z4]  642:   Addr_Block.Pts_Out_point :=
[9F2]  643:     Addr(Pts_Out);
[LC4]  644:   Regs.DS := Seg(Addr_Block);
[9E4]  645:   Regs.DX := Ofs(Addr_Block);
[8V5]  646:   Regs.CX := 1139; { VDI - Kennziffer }
[AF2]  647:   Intr($EF,Regs);
[ZNØ]  648: end;
[ØØØ]  649:
[5U4]  65Ø: function Init_GEM: Integer;
[ZH2]  651: var i: Integer;
[MUØ]  652: begin
[ME2]  653:   { appl_init }
[H26]  654:   AES_Call(1Ø,Int_In,Int_Out,Addr_In,
[6S1]  655:     Addr_Out);
[Ø64]  656:   Init_GEM := Int_Out[Ø];
[2W2]  657:   { graf_handle }
[Z36]  658:   AES_Call(77,Int_In,Int_Out,Addr_In,
[6S1]  659:     Addr_Out);
[T64]  66Ø:   VDI_Handle := Int_Out[Ø];
[KC5]  661:   { virtuelle workstation auf }
[PG2]  662:   for i := 1 to 9 do
[UK2]  663:   Int_In[i] := 1;
[KK2]  664:   Int_In[1Ø] := 2;
[HL5]  665:   VDI_call(1ØØ,Ø,11,Ø,Control,Int_In,
[9G5]  666:             Int_Out,Pts_In,Pts_Out,False);
[354]  667:   VDI_Handle := Control[6];
[AZ3]  668:   GEMResX := Int_Out[Ø];
[FZ3]  669:   GEMResY := Int_Out[1];
[ZNØ]  67Ø: end;
[ØØØ]  671:
[NG3]  672: procedure Exit_GEM;
[MUØ]  673: begin
[4F2]  674:   { appl_exit }
[N36]  675:   AES_Call(19,Int_In,Int_Out,Addr_In,
[6S1]  676:     Addr_Out);
[KA5]  677:   { close virtual workstation }
[TA5]  678:   VDI_call(1Ø1,Ø,Ø,Ø,Control,Int_In,
[9G5]  679:             Int_Out,Pts_In,Pts_Out,False);
[ZNØ]  68Ø: end;
[ØØØ]  681:
[8T3]  682: procedure Hide_Mouse;
[MUØ]  683: begin
[3B5]  684:   VDI_call(123,Ø,Ø,Ø,Control,Int_In,
[9G5]  685:             Int_Out,Pts_In,Pts_Out,False);
[ZNØ]  686: end;
[ØØØ]  687:
[ZW3]  688: procedure Show_Mouse;
[MUØ]  689: begin
[1K2]  69Ø:   Int_In[Ø] := 1;
[3B5]  691:   VDI_call(122,Ø,1,Ø,Control,Int_In,
[9G5]  692:             Int_Out,Pts_In,Pts_Out,False);
[ZNØ]  693: end;
[ØØØ]  694:
[MF7]  695: procedure Set_Mouse (M_Type: Mouse_Type);
[MUØ]  696: begin
[RC4]  697:   Int_In[Ø] := Ord(M_Type);
[346]  698:   AES_Call(78,Int_In,Int_Out,Addr_In,
[6S1]  699:           Addr_Out);
[ZNØ]  7ØØ: end;
[ØØØ]  7Ø1:
[8V3]  7Ø2: procedure Init_Mouse;
[MUØ]  7Ø3: begin
[YH2]  7Ø4:   Int_In[Ø] := Ø;
[2B5]  7Ø5:   VDI_call(122,Ø,Ø,Ø,Control,Int_In,
[9G5]  7Ø6:             Int_Out,Pts_In,Pts_Out,False);
[BG3]  7Ø7:   Set_Mouse(M_Arrow);
[ZNØ]  7Ø8: end;
[ØØØ]  7Ø9:
[MC5]  71Ø: function Do_Alert (maske: Str255;
[6W3]  711:           btn: Integer): Integer;
[MUØ]  712: begin
[WW2]  713:   Int_In[Ø] := btn;
[B55]  714:   maske[Length(maske)+1] := Chr(Ø);
[KT2]  715:   { C-Konvention }
[2R4]  716:   Addr_In[Ø] := Addr(maske[1]);
[WL4]  717:   { Ø. Byte überspringen ! }
[536]  718:   AES_Call(52,Int_In,Int_Out,Addr_In,
[6S1]  719:           Addr_Out);
[D84]  72Ø:   Do_Alert := Int_Out[Ø];
[ZNØ]  721: end;
[ØØØ]  722:
[M34]  723: function Load_Resource
[GV3]  724:     (name: string): Boolean;
[MUØ]  725: begin
[9Ø5]  726:   name[Length(name)+1] := Chr(Ø);
[BE4]  727:   Addr_In[Ø] := Addr(name[1]);
[5A6]  728:   AES_Call(11Ø,Int_In,Int_Out,Addr_In,
[6S1]  729:           Addr_Out);
[8U5]  73Ø:   Load_Resource := (Int_Out[Ø] <> Ø);
[ZNØ]  731: end;
[ØØØ]  732:
[WA4]  733: procedure Free_Resource;
[MUØ]  734: begin
[6A6]  735:   AES_Call(111,Int_In,Int_Out,Addr_In,
[6S1]  736:           Addr_Out);
[ZNØ]  737: end;
[ØØØ]  738:
[7D6]  739: procedure Find_Menu (Index: Integer;
[YF3]  74Ø:           var Menu: Menu_Ptr);
[ØT1]  741: var Trick:
[Y11]  742: record
[BM2]  743:   case Integer of
[NL2]  744:     Ø: (M: Menu_Ptr);
[U92]  745:     1: (p: Pointer);
[ZNØ]  746:   end;
[MUØ]  747:   begin
[AK3]  748:     Int_In[Ø] := R_Tree;
[SC3]  749:     Int_In[1] := Index;
[7A6]  75Ø:     AES_Call(112,Int_In,Int_Out,Addr_In,
[6S1]  751:              Addr_Out);
[RØ4]  752:     Trick.p := Addr_Out[Ø];
[CM2]  753:     Menu := Trick.M;
[ZNØ]  754:   end;
[ØØØ]  755:
[3D6]  756: procedure Find_Dialog(Index: Integer;
[LR3]  757:           var Dial: Dialog_Ptr);
[ØT1]  758: var Trick:
[Y11]  759: record
[BM2]  76Ø:   case Integer of
[EY2]  761:     Ø: (d: Dialog_Ptr);
[U92]  762:     1: (p: Pointer);
[ZNØ]  763:   end;
[MUØ]  764:   begin
[AK3]  765:     Int_In[Ø] := R_Tree;
[SC3]  766:     Int_In[1] := Index;
[7A6]  767:     AES_Call(112,Int_In,Int_Out,Addr_In,
[6S1]  768:       Addr_Out);
[RØ4]  769:     Trick.p := Addr_Out[Ø];
[5H2]  77Ø:     Dial := Trick.d;
[ZNØ]  771:   end;
[ØØØ]  772:
[C57]  773: procedure form_dial (flag,litx,lity,litw,
[LC3]  774:           lith,bigx,bigy,bigw,
[CF2]  775:           bigh: Integer);
[MUØ]  776: begin
[423]  777:   Int_In[Ø] := flag;
[Ø43]  778:   Int_In[1] := litx;
[343]  779:   Int_In[2] := lity;
[Ø43]  78Ø:   Int_In[3] := litw;
[333]  781:   Int_In[4] := lith;
[T23]  782:   Int_In[5] := bigx;
[W23]  783:   Int_In[6] := bigy;
[T23]  784:   Int_In[7] := bigw;
[W13]  785:   Int_In[8] := bigh;
[B96]  786:   AES_Call (51,Int_In,Int_Out,Addr_In,
[6S1]  787:            Addr_Out);
[ZNØ]  788: end;
[ØØØ]  789:
[ZC6]  79Ø: function form_Do (dialog: Dialog_Ptr;
[2V4]  791:           ed_start: Integer): Integer;
[MUØ]  792: begin
[HU3]  793:   Int_In[Ø] := ed_start;
[S76]  794:   Addr_In[Ø] := dptr_to_point(dialog);
[A96]  795:   AES_Call (5Ø,Int_In,Int_Out,Addr_In,
[6S1]  796:            Addr_Out);
[SZ3]  797:   form_Do := Int_Out[Ø];
[ZNØ]  798: end;
[ØØØ]  799:
[537]  8ØØ: procedure Objc_Draw (dialog: Dialog_Ptr;
[SØ5]  8Ø1:           start,tiefe,x,y,w,h: Integer);
[MUØ]  8Ø2: begin
[SE3]  8Ø3:   Int_In[Ø] := start;
[FB3]  8Ø4:   Int_In[1] := tiefe;
[SN2]  8Ø5:   Int_In[2] := x;
[WN2]  8Ø6:   Int_In[3] := y;
[RN2]  8Ø7:   Int_In[4] := w;
[BM2]  8Ø8:   Int_In[5] := h;
[S76]  8Ø9:   Addr_In[Ø] := dptr_to_point(dialog);
[896]  81Ø:   AES_Call (42,Int_In,Int_Out,Addr_In,
[6S1]  811:            Addr_Out);
[ZNØ]  812: end;
[ØØØ]  813:
[KT6]  814: function Do_dialog (dialog: Dialog_Ptr;
[2V4]  815:           ed_start: Integer): Integer;
[D96]  816: var xdial,ydial,wdial,hdial: Integer;
[DW4]  817:     oldx,oldy,oldw,oldh: Integer;
[MUØ]  818: begin
[7X4]  819:   xdial := dialog^[Root].ob_x;
[DX4]  82Ø:   ydial := dialog^[Root].ob_y;
[1X4]  821:   wdial := dialog^[Root].ob_w;
[7U4]  822:   hdial := dialog^[Root].ob_h;
[8Ø5]  823:   if dialog^[Root].ob_state AND
[5Z2]  824:      Outlined <> Ø then
[MUØ]  825:   begin
[RV2]  826:     xdial := xdial - 3;
[A64]  827:     if xdial < Ø then xdial := Ø;
[WV2]  828:     ydial := ydial - 3;
[F64]  829:     if ydial < Ø then ydial := Ø;
[TV2]  83Ø:     wdial := wdial + 6;
[ET2]  831:     hdial := hdial + 6;
[ZNØ]  832:   end;
[R22]  833:   Hide_Mouse;
```

Listing 1. Die Unit »gembind.pas« ist die Schnittstelle zwischen Turbo Pascal und GEM

```
[WZ5]  834:   form_dial (Ø,Ø,Ø,Ø,Ø,xdial,ydial,wdial,
[Ø61]  835:             hdial);
[LS6]  836:   Objc_Draw (dialog,Root,Max_Depth,xdial,
[G83]  837:             ydial,wdial,hdial);
[852]  838:   Show_Mouse;
[9L6]  839:   Do_dialog := form_Do (dialog,ed_start);
[ZZ5]  84Ø:   form_dial (3,Ø,Ø,Ø,Ø,xdial,ydial,wdial,
[Ø61]  841:             hdial);
[ZNØ]  842: end;
[ØØØ]  843:
[AC7]  844: procedure Show_Dialog (dialog: Dialog_Ptr);
[MUØ]  845: begin
[W43]  846:   Int_In[Ø] := Root;
[Ø34]  847:   Int_In[1] := Max_Depth;
[ØK2]  848:   Int_In[2] := Ø;
[1K2]  849:   Int_In[3] := Ø;
[9M3]  85Ø:   Int_In[4] := GEMResX;
[BM3]  851:   Int_In[5] := GEMResY;
[S76]  852:   Addr_In[Ø] := dptr_to_point(dialog);
[896]  853:   AES_Call (42,Int_In,Int_Out,Addr_In,
[6S1]  854:     Addr_Out);
[ZNØ]  855: end;
[ØØØ]  856:
[3D7]  857: procedure Center_Dialog (Dial: Dialog_Ptr);
[MUØ]  858: begin
[9X5]  859:   Addr_In[Ø] := dptr_to_point(Dial);
[D36]  86Ø:   AES_Call(54,Int_In,Int_Out,Addr_In,
[6S1]  861:            Addr_Out);
[ZNØ]  862: end;
[ØØØ]  863:
[757]  864: procedure End_dialog (dialog: Dialog_Ptr);
[D96]  865: var xdial,ydial,wdial,hdial: Integer;
[MUØ]  866: begin
[7X4]  867:   xdial := dialog^[Root].ob_x;
[DX4]  868:   ydial := dialog^[Root].ob_y;
[1X4]  869:   wdial := dialog^[Root].ob_w;
[7U4]  87Ø:   hdial := dialog^[Root].ob_h;
[8Ø5]  871:   if dialog^[Root].ob_state AND
[5Z2]  872:      Outlined <> Ø then
[MUØ]  873:   begin
[RV2]  874:     xdial := xdial - 3;
[A64]  875:     if xdial < Ø then xdial := Ø;
[WV2]  876:     ydial := ydial - 3;
[F64]  877:     if ydial < Ø then ydial := Ø;
[TV2]  878:     wdial := wdial + 6;
[ET2]  879:     hdial := hdial + 6;
[ZNØ]  88Ø:   end;
[R22]  881:   Hide_Mouse;
[ZZ5]  882:   form_dial (3,Ø,Ø,Ø,Ø,xdial,ydial,wdial,
[Ø61]  883:             hdial);
[852]  884:   Show_Mouse;
[ZNØ]  885: end;
[ØØØ]  886:
[AN6]  887: procedure Set_DText (Dial: Dialog_Ptr;
[ØN4]  888:           item: Tree_Index; s: Str255;
[M25]  889: font: Integer; just: TE_Just);
[VK3]  89Ø: var C_str: C_String;
[8V1]  891:     i: Integer;
[MUØ]  892: begin
[ØF3]  893:   Pas_To_C (s,C_str);
[LYØ]  894:   i := Ø;
[KL6]  895:   if (Dial^[item].Ob_Type = G_String) OR
[N66]  896:      (Dial^[item].Ob_Type = G_Title) then
[MUØ]  897:   begin
[KB4]  898:     while C_str[i] <> Chr(Ø) do
[MUØ]  899:     begin
[ZL5]  9ØØ:       Dial^[item].ob_spec.Str^[i] :=
[FR1]  9Ø1:         C_str[i];
[WC1]  9Ø2:       i := i + 1;
[ZNØ]  9Ø3:     end;
[FW1]  9Ø4:     if s = '' then
[ZL5]  9Ø5:       Dial^[item].ob_spec.Str^[i] :=
[N41]  9Ø6:         Chr(Ø);
[ZNØ]  9Ø7:   end;
[176]  9Ø8:   if (Dial^[item].Ob_Type = G_Text) OR
[T86]  9Ø9:      (Dial^[item].Ob_Type = G_BoxText) OR
[ZX5]  91Ø:      (Dial^[item].Ob_Type = G_FText) OR
[SX5]  911:      (Dial^[item].Ob_Type = G_FBoxText)
[2RØ]  912:      then
[MUØ]  913:   begin
[KB4]  914:     while C_str[i] <> Chr(Ø) do
[MUØ]  915:     begin
[GS4]  916:       Dial^[item].ob_spec.info^.
[DN4]  917:         te_ptext^[i] := C_str[i];
[WC1]  918:       i := i + 1;
[ZNØ]  919:     end;
[FW1]  92Ø:     if s = '' then
[GS4]  921:       Dial^[item].ob_spec.info^.
[L24]  922:         te_ptext^[i] := Chr(Ø);
[CF6]  923:     Dial^[item].ob_spec.info^.te_font :=
[VVØ]  924:       font;
[NG6]  925:     Dial^[item].ob_spec.info^.TE_Just :=
[2S1]  926:       Ord(just);
[ZNØ]  927:   end;
[ZNØ]  928: end;
[ØØØ]  929:
[V47]  93Ø: procedure Obj_SetState (Dial: Dialog_Ptr;
[VØ3]  931:           Index: Tree_Index;
[VF5]  932:           state: Integer; redraw: Boolean);
[MUØ]  933: begin
```

```
[CF5]  934:   Dial^[Index].ob_state := state;
[YD5]  935:   if redraw then Show_Dialog (Dial);
[ZNØ]  936: end;
[ØØØ]  937:
[8C6]  938: function Obj_State (Dial: Dialog_Ptr;
[B14]  939:          Index: Integer): Integer;
[MUØ]  94Ø: begin
[D86]  941:   Obj_State := Dial^[Index].ob_state;
[ZNØ]  942: end;
[ØØØ]  943:
[WH6]  944: procedure Get_DEdit (Dial: Dialog_Ptr;
[1D5]  945:           item: Tree_Index; var s: Str255);
[VK3]  946: var C_str: C_String;
[MUØ]  947: begin
[CB1]  948:   C_str :=
[7U6]  949:     Dial^[item].ob_spec.info^.te_ptext^;
[DC3]  95Ø:   C_to_pas (C_str,s);
[ZNØ]  951: end;
[ØØØ]  952:
[XF6]  953: procedure Draw_Menu (Menu: Menu_Ptr);
[MUØ]  954: begin
[GØ6]  955:   Addr_In[Ø] := Mptr_to_point(Menu);
[1K2]  956:   Int_In[Ø] := 1;
[R26]  957:   AES_Call(3Ø,Int_In,Int_Out,Addr_In,
[6S1]  958:     Addr_Out);
[ZNØ]  959: end;
[ØØØ]  96Ø:
[2L6]  961: procedure Erase_Menu (Menu: Menu_Ptr);
[MUØ]  962: begin
[GØ6]  963:   Addr_In[Ø] := Mptr_to_point(Menu);
[YH2]  964:   Int_In[Ø] := Ø;
[R26]  965:   AES_Call(3Ø,Int_In,Int_Out,Addr_In,
[6S1]  966:     Addr_Out);
[ZNØ]  967: end;
[ØØØ]  968:
[NM6]  969: procedure Menu_Enable (Menu: Menu_Ptr;
[U23]  97Ø:           item: Tree_Index);
[MUØ]  971: begin
[GØ6]  972:   Addr_In[Ø] := Mptr_to_point(Menu);
[X33]  973:   Int_In[Ø] := item;
[2K2]  974:   Int_In[1] := 1;
[Z26]  975:   AES_Call(32,Int_In,Int_Out,Addr_In,
[6S1]  976:           Addr_Out);
[ZNØ]  977: end;
[ØØØ]  978:
[3Z6]  979: procedure Menu_Disable (Menu: Menu_Ptr;
[U23]  98Ø:           item: Tree_Index);
[MUØ]  981: begin
[GØ6]  982:   Addr_In[Ø] := Mptr_to_point(Menu);
[X33]  983:   Int_In[Ø] := item;
[ZH2]  984:   Int_In[1] := Ø;
[Z26]  985:   AES_Call(32,Int_In,Int_Out,Addr_In,
[6S1]  986:     Addr_Out);
[ZNØ]  987: end;
[ØØØ]  988:
[2S6]  989: procedure Menu_Normal (Menu: Menu_Ptr;
[T73]  99Ø:           Title: Tree_Index);
[MUØ]  991: begin
[GØ6]  992:   Addr_In[Ø] := Mptr_to_point(Menu);
[9C3]  993:   Int_In[Ø] := Title;
[2K2]  994:   Int_In[1] := 1;
[336]  995:   AES_Call(33,Int_In,Int_Out,Addr_In,
[6S1]  996:     Addr_Out);
[ZNØ]  997: end;
[ØØØ]  998:
[WØ7]  999: procedure Menu_Hilight (Menu: Menu_Ptr;
[T73] 1ØØØ:           Title: Tree_Index);
[MUØ] 1ØØ1: begin
[GØ6] 1ØØ2:   Addr_In[Ø] := Mptr_to_point(Menu);
[9C3] 1ØØ3:   Int_In[Ø] := Title;
[ZH2] 1ØØ4:   Int_In[1] := Ø;
[336] 1ØØ5:   AES_Call(33,Int_In,Int_Out,Addr_In,
[6S1] 1ØØ6:     Addr_Out);
[ZNØ] 1ØØ7: end;
[ØØØ] 1ØØ8:
[5P6] 1ØØ9: function New_Window (w_type: Integer;
[Z84] 1Ø1Ø:           var Title: Window_Title;
[PC3] 1Ø11:           x_max,y_max,w_max,
[W34] 1Ø12:           h_max: Integer): Integer;
[MUØ] 1Ø13: begin
[8M3] 1Ø14:   Int_In[Ø] := w_type;
[AG3] 1Ø15:   Int_In[1] := x_max;
[EG3] 1Ø16:   Int_In[2] := y_max;
[9G3] 1Ø17:   Int_In[3] := w_max;
[XE3] 1Ø18:   Int_In[4] := h_max;
[1A6] 1Ø19:   AES_Call(1ØØ,Int_In,Int_Out,Addr_In,
[6S1] 1Ø2Ø:     Addr_Out);
[PB4] 1Ø21:   New_Window := Int_Out[Ø];
[HB4] 1Ø22:   Int_In[Ø] := Int_Out[Ø];
[DM3] 1Ø23:   Int_In[1] := WF_Name;
[E85] 1Ø24:   Title[Length(Title)+1] := Chr(Ø);
[FB6] 1Ø25:   split_addr (Addr(Title[1]),Int_In[2],
[732] 1Ø26:               Int_In[3]);
[6A6] 1Ø27:   AES_Call(1Ø5,Int_In,Int_Out,Addr_In,
[6S1] 1Ø28:           Addr_Out);
[ZNØ] 1Ø29: end;
[ØØØ] 1Ø3Ø:
[7P6] 1Ø31: procedure Open_window (Handle: Integer;
[ZX2] 1Ø32:           x,y,w,h: Integer);
[MUØ] 1Ø33: begin
```

```
[7D3] 1Ø34:   Int_In[Ø] := Handle;
[RN2] 1Ø35:   Int_In[1] := x;
[VN2] 1Ø36:   Int_In[2] := y;
[PN2] 1Ø37:   Int_In[3] := w;
[AM2] 1Ø38:   Int_In[4] := h;
[2A6] 1Ø39:   AES_Call(1Ø1,Int_In,Int_Out,Addr_In,
[6S1] 1Ø4Ø:           Addr_Out);
[ZNØ] 1Ø41: end;
[ØØØ] 1Ø42:
[ED6] 1Ø43: procedure Set_WName (Handle: Integer;
[V44] 1Ø44:           var name: Window_Title);
[MUØ] 1Ø45: begin
[7D3] 1Ø46:   Int_In[Ø] := Handle;
[DM3] 1Ø47:   Int_In[1] := WF_Name;
[9Ø5] 1Ø48:   name[Length(name)+1] := Chr(Ø);
[456] 1Ø49:   split_addr (Addr(name[1]),Int_In[2],
[732] 1Ø5Ø:             Int_In[3]);
[6A6] 1Ø51:   AES_Call(1Ø5,Int_In,Int_Out,Addr_In,
[6S1] 1Ø52:           Addr_Out);
[ZNØ] 1Ø53: end;
[ØØØ] 1Ø54:
[PE6] 1Ø55: procedure Set_Winfo (Handle: Integer;
[654] 1Ø56:           var info: Window_Title);
[MUØ] 1Ø57: begin
[7D3] 1Ø58:   Int_In[Ø] := Handle;
[SM3] 1Ø59:   Int_In[1] := WF_Info;
[ZØ5] 1Ø6Ø:   info[Length(info)+1] := Chr(Ø);
[P66] 1Ø61:   split_addr (Addr(info[1]),Int_In[2],
[732] 1Ø62:             Int_In[3]);
[6A6] 1Ø63:   AES_Call(1Ø5,Int_In,Int_Out,Addr_In,
[6S1] 1Ø64:           Addr_Out);
[ZNØ] 1Ø65: end;
[ØØØ] 1Ø66:
[Ø27] 1Ø67: procedure Close_Window (Handle: Integer);
[MUØ] 1Ø68: begin
[7D3] 1Ø69:   Int_In[Ø] := Handle;
[3A6] 1Ø7Ø:   AES_Call(1Ø2,Int_In,Int_Out,Addr_In,
[6S1] 1Ø71:           Addr_Out);
[ZNØ] 1Ø72: end;
[ØØØ] 1Ø73:
[N87] 1Ø74: procedure Delete_Window (Handle: Integer);
[MUØ] 1Ø75: begin
[7D3] 1Ø76:   Int_In[Ø] := Handle;
[4A6] 1Ø77:   AES_Call(1Ø3,Int_In,Int_Out,Addr_In,
[6S1] 1Ø78:           Addr_Out);
[ZNØ] 1Ø79: end;
[ØØØ] 1Ø8Ø:
[KE6] 1Ø81: procedure Work_Rect (Handle: Integer;
[PM3] 1Ø82:           var x,y,w,h: Integer);
[MUØ] 1Ø83: begin
[7D3] 1Ø84:   Int_In[Ø] := Handle;
[2K4] 1Ø85:   Int_In[1] := WF_WorkXYWH;
[5A6] 1Ø86:   AES_Call(1Ø4,Int_In,Int_Out,Addr_In,
[6S1] 1Ø87:           Addr_Out);
[7T2] 1Ø88:   x := Int_Out[1];
[CT2] 1Ø89:   y := Int_Out[2];
[BT2] 1Ø9Ø:   w := Int_Out[3];
[GS2] 1Ø91:   h := Int_Out[4];
[ZNØ] 1Ø92: end;
[ØØØ] 1Ø93:
[ZP4] 1Ø94: procedure Wind_Get (Handle,
[VZ2] 1Ø95:           request: Integer;
[234] 1Ø96:           var v1,v2,v3,v4: Integer);
[MUØ] 1Ø97: begin
[7D3] 1Ø98:   Int_In[Ø] := Handle;
[DN3] 1Ø99:   Int_In[1] := request;
[5A6] 11ØØ:   AES_Call(1Ø4,Int_In,Int_Out,Addr_In,
[6S1] 11Ø1:           Addr_Out);
[BR2] 11Ø2:   v1 := Int_Out[1];
[KR2] 11Ø3:   v2 := Int_Out[2];
[TR2] 11Ø4:   v3 := Int_Out[3];
[ØS2] 11Ø5:   v4 := Int_Out[4];
[ZNØ] 11Ø6: end;
[ØØØ] 11Ø7:
[BR4] 11Ø8: procedure Wind_Set (Handle,
[VZ2] 11Ø9:           request: Integer;
[BC3] 111Ø:           v1,v2,v3,v4: Integer);
[MUØ] 1111: begin
[7D3] 1112:   Int_In[Ø] := Handle;
[DN3] 1113:   Int_In[1] := request;
[5P2] 1114:   Int_In[2] := v1;
[AP2] 1115:   Int_In[3] := v2;
[FP2] 1116:   Int_In[4] := v3;
[MP2] 1117:   Int_In[5] := v4;
[6A6] 1118:   AES_Call(1Ø5,Int_In,Int_Out,Addr_In,
[6S1] 1119:           Addr_Out);
[ZNØ] 112Ø: end;
[ØØØ] 1121:
[6M5] 1122: function Front_Window: Integer;
[MUØ] 1123: begin
[YH2] 1124:   Int_In[Ø] := Ø;
[FK3] 1125:   Int_In[1] := WF_Top;
[5A6] 1126:   AES_Call(1Ø4,Int_In,Int_Out,Addr_In,
[6S1] 1127:           Addr_Out);
[B25] 1128:   Front_Window := Int_Out[1];
[ZNØ] 1129: end;
[ØØØ] 113Ø:
[GS6] 1131: procedure Set_Window (Handle: Integer);
[MUØ] 1132: begin
[7D3] 1133:   Int_In[Ø] := Handle;
[FK3] 1134:   Int_In[1] := WF_Top;
```

```
[6A6] 1135:   AES_Call(1Ø5,Int_In,Int_Out,Addr_In,
[6S1] 1136:           Addr_Out);
[ZNØ] 1137: end;
[ØØØ] 1138:
[2D5] 1139: function Get_in_file (var Path,
[CB4] 114Ø:          name: Path_Name): Boolean;
[ZH2] 1141: var i: Integer;
[TE3] 1142:     connect: Path_Name;
[MUØ] 1143: begin
[F25] 1144:   Path[Length(Path)+1] := Chr(Ø);
[EF4] 1145:   Addr_In[Ø] := Addr(Path[1]);
[9Ø5] 1146:   name[Length(name)+1] := Chr(Ø);
[DE4] 1147:   Addr_In[1] := Addr(name[1]);
[936] 1148:   AES_Call(9Ø,Int_In,Int_Out,Addr_In,
[6S1] 1149:           Addr_Out);
[AD5] 115Ø:   Get_in_file := (Int_Out[1] <> Ø);
[PU3] 1151:   if Int_Out[1] <> Ø then
[MUØ] 1152:   begin
[PYØ] 1153:     i := 1;
[K34] 1154:     while Path[i] <> Chr(Ø) do
[WC1] 1155:     i := i + 1;
[YZ2] 1156:     Path[Ø] := Chr(i-1);
[PYØ] 1157:     i := 1;
[L24] 1158:     while name[i] <> Chr(Ø) do
[WC1] 1159:     i := i + 1;
[VY2] 116Ø:     name[Ø] := Chr(i-1);
[GM2] 1161:     connect := Path;
[P31] 1162:     repeat
[ME5] 1163:       Delete(connect,Length(connect),1);
[A76] 1164:     until connect[Length(connect)] = '\';
[465] 1165:     connect := Concat(connect,name);
[HN2] 1166:     name := connect;
[ZNØ] 1167:   end;
[ZNØ] 1168: end;
[ØØØ] 1169:
[N66] 117Ø: procedure Dragbox (x1,y1,w1,h1,x2,y2,w2,
[PL3] 1171:           h2: Integer; var Endx,
[6G2] 1172:           Endy: Integer);
[MUØ] 1173: begin
[AP2] 1174:   Int_In[Ø] := w2;
[YM2] 1175:   Int_In[1] := h2;
[FP2] 1176:   Int_In[2] := x2;
[LP2] 1177:   Int_In[3] := y2;
[DP2] 1178:   Int_In[4] := x1;
[HP2] 1179:   Int_In[5] := y1;
[CP2] 118Ø:   Int_In[6] := w1;
[ØN2] 1181:   Int_In[7] := h1;
[736] 1182:   AES_Call(71,Int_In,Int_Out,Addr_In,
[6S1] 1183:           Addr_Out);
[PD3] 1184:   Endx := Int_Out[1];
[TD3] 1185:   Endy := Int_Out[2];
[ZNØ] 1186: end;
[ØØØ] 1187:
[ZH6] 1188: procedure Growbox (stx,sty,stw,sth,fix,
[PS3] 1189:           fiy,fiw,fih: Integer);
[MUØ] 119Ø: begin
[TY2] 1191:   Int_In[Ø] := stx;
[VY2] 1192:   Int_In[1] := sty;
[UY2] 1193:   Int_In[2] := stw;
[CY2] 1194:   Int_In[3] := sth;
[AW2] 1195:   Int_In[4] := fix;
[CW2] 1196:   Int_In[5] := fiy;
[BW2] 1197:   Int_In[6] := fiw;
[XV2] 1198:   Int_In[7] := fih;
[N96] 1199:   AES_Call (73,Int_In,Int_Out,Addr_In,
[6S1] 12ØØ:           Addr_Out);
[ZNØ] 12Ø1: end;
[ØØØ] 12Ø2:
[CA7] 12Ø3: procedure Shrinkbox (stx,sty,stw,sth,fix,
[PS3] 12Ø4:           fiy,fiw,fih: Integer);
[MUØ] 12Ø5: begin
[TY2] 12Ø6:   Int_In[Ø] := stx;
[VY2] 12Ø7:   Int_In[1] := sty;
[UY2] 12Ø8:   Int_In[2] := stw;
[CY2] 12Ø9:   Int_In[3] := sth;
[AW2] 121Ø:   Int_In[4] := fix;
[CW2] 1211:   Int_In[5] := fiy;
[BW2] 1212:   Int_In[6] := fiw;
[XV2] 1213:   Int_In[7] := fih;
[P96] 1214:   AES_Call (74,Int_In,Int_Out,Addr_In,
[6S1] 1215:           Addr_Out);
[ZNØ] 1216: end;
[ØØØ] 1217:
[TT6] 1218: function Get_Event (emask,bmask,bstate,
[Y23] 1219:          n_clicks: Integer;
[BL2] 122Ø:          ticks: Longint;
[ØV2] 1221:          m1_flag: Boolean;
[M44] 1222:          m1x,m1y,m1w,m1h: Integer;
[3V2] 1223:          m2_flag: Boolean;
[Ø54] 1224:          m2x,m2y,m2w,m2h: Integer;
[SY4] 1225:          var message: Message_Buffer;
[2F4] 1226:          var key,brtn,bclick,mx,my,
[HF4] 1227:          kstate: Integer): Integer;
[MUØ] 1228: begin
[TB3] 1229:   Int_In[Ø] := emask;
[DW3] 123Ø:   Int_In[1] := n_clicks;
[KB3] 1231:   Int_In[2] := bmask;
[SF3] 1232:   Int_In[3] := bstate;
```

Listing 1. Die Unit »gembind.pas« ist die Schnittstelle zwischen Turbo Pascal und GEM

```
[RA4] 1233:   Int_In[4] := Ord(m1_flag);
[ØU2] 1234:   Int_In[5] := m1x;
[2U2] 1235:   Int_In[6] := m1y;
[1U2] 1236:   Int_In[7] := m1w;
[LT2] 1237:   Int_In[8] := m1h;
[ØB4] 1238:   Int_In[9] := Ord(m2_flag);
[PØ3] 1239:   Int_In[1Ø] := m2x;
[UØ3] 124Ø:   Int_In[11] := m2y;
[SØ3] 1241:   Int_In[12] := m2w;
[WZ2] 1242:   Int_In[13] := m2h;
[6F4] 1243:   Addr_In[Ø] := Addr(message);
[115] 1244:   Int_In[14] := ticks MOD $1ØØØØ;
[725] 1245:   Int_In[15] := ticks DIV $1ØØØØ;
[836] 1246:   AES_Call(25,Int_In,Int_Out,Addr_In,
[6S1] 1247:           Addr_Out);
[EU2] 1248:   mx := Int_Out[1];
[NU2] 1249:   my := Int_Out[2];
[K73] 125Ø:   key := Int_Out[5];
[WF3] 1251:   brtn := Int_Out[6];
[1F3] 1252:   bclick := Int_Out[3];
[FK3] 1253:   kstate := Int_Out[4];
[P84] 1254:   Get_Event := Int_Out[Ø];
[ZNØ] 1255: end;
[ØØØ] 1256:
[KC6] 1257: procedure Set_Clip (x,y,w,h: Integer);
[MUØ] 1258: begin
[2Z3] 1259:   Int_In [Ø] := 1; { True }
[4K2] 126Ø:   Pts_In [Ø] := x;
[AK2] 1261:   Pts_In [1] := y;
[SG3] 1262:   Pts_In [2] := x + w - 1;
[2F3] 1263:   Pts_In [3] := y + h - 1;
[PB5] 1264:   VDI_call(129,Ø,1,4,Control,Int_In,
[9G5] 1265:           Int_Out,Pts_In,Pts_Out,False);
[ZNØ] 1266: end;
[ØØØ] 1267:
[X66] 1268: procedure Draw_String(x,y: Integer;
[LM1] 1269:           s: Str255);
[ZH2] 127Ø: var i: Integer;
[MUØ] 1271: begin
[KP2] 1272:   Pts_In[Ø] := x;
[PP2] 1273:   Pts_In[1] := y;
[PYØ] 1274:   i := 1;
[TK3] 1275:   while (i <= Length(s)) do
[MUØ] 1276:   begin
[Y64] 1277:     Int_In[i-1] := Ord(s[i]);
[WC1] 1278:     i := i + 1;
[ZNØ] 1279:   end;
[RK2] 128Ø:   Int_In[i] := Ø;
[RM6] 1281:   VDI_call(8,Ø,Length(s),1,Control,Int_In,
[9G5] 1282:           Int_Out,Pts_In,Pts_Out,False);
[ZNØ] 1283: end;
[ØØØ] 1284:
[EN3] 1285: procedure Line_Color
[RZ3] 1286:           (color_index: Integer);
[MUØ] 1287: begin
[6A4] 1288:   Int_In [Ø] := color_index;
[WS6] 1289:   VDI_call(17,Ø,1,Ø,Control,Int_In,Int_Out,
[5Y3] 129Ø:           Pts_In,Pts_Out,False);
[ZNØ] 1291: end;
[ØØØ] 1292:
[N26] 1293: procedure Draw_Mode (mode: Integer);
[MUØ] 1294: begin
[L23] 1295:   Int_In [Ø] := mode;
[ES6] 1296:   VDI_call(32,Ø,1,Ø,Control,Int_In,Int_Out,
[5Y3] 1297:           Pts_In,Pts_Out,False);
[ZNØ] 1298: end;
[ØØØ] 1299:
[XT3] 13ØØ: procedure Paint_Color
[RZ3] 13Ø1:           (color_index: Integer);
[MUØ] 13Ø2: begin
[6A4] 13Ø3:   Int_In [Ø] := color_index;
[RS6] 13Ø4:   VDI_call(25,Ø,1,Ø,Control,Int_In,Int_Out,
[5Y3] 13Ø5:           Pts_In,Pts_Out,False);
[ZNØ] 13Ø6: end;
[ØØØ] 13Ø7:
[3W3] 13Ø8: procedure Paint_Style
[VØ4] 13Ø9:           (style_index: Integer);
[MUØ] 131Ø: begin
[GC2] 1311:   Int_In [Ø] := 2;
[FS6] 1312:   VDI_call(23,Ø,1,Ø,Control,Int_In,Int_Out,
[5Y3] 1313:           Pts_In,Pts_Out,False);
[8B4] 1314:   Int_In [Ø] := style_index;
[LS6] 1315:   VDI_call(24,Ø,1,Ø,Control,Int_In,Int_Out,
[5Y3] 1316:           Pts_In,Pts_Out,False);
[ZNØ] 1317: end;
[ØØØ] 1318:
[NP6] 1319: procedure Paint_Rect (x,y,w,h: Integer);
[MUØ] 132Ø: begin
[KP2] 1321:   Pts_In[Ø] := x;
[PP2] 1322:   Pts_In[1] := y;
[4K3] 1323:   Pts_In[2] := x + w - 1;
[VG3] 1324:   Pts_In[3] := y + h - 1;
[AS6] 1325:   VDI_call(11,1,Ø,4,Control,Int_In,Int_Out,
[5Y3] 1326:           Pts_In,Pts_Out,False);
[ZNØ] 1327: end;
[ØØØ] 1328:
[UA6] 1329: procedure Line_Endstyle (beg_style,
[8C3] 133Ø:           End_style: Integer);
[MUØ] 1331: begin
[AØ4] 1332:   Int_In [Ø] := beg_style;
```

```
[VØ4] 1333:   Int_In [1] := End_style;
[2B5] 1334:   VDI_call(1Ø8,Ø,2,Ø,Control,Int_In,
[9G5] 1335:           Int_Out,Pts_In,Pts_Out,False);
[ZNØ] 1336: end;
[ØØØ] 1337:
[KA7] 1338: procedure Line_Style (style: Line_Types);
[MUØ] 1339: begin
[K83] 134Ø:   Int_In [Ø] := style;
[MS6] 1341:   VDI_call(15,Ø,1,Ø,Control,Int_In,Int_Out,
[5Y3] 1342:           Pts_In,Pts_Out,False);
[ZNØ] 1343: end;
[ØØØ] 1344:
[LB6] 1345: procedure Pline (x1,y1,x2,y2: Integer);
[MUØ] 1346: begin
[5R2] 1347:   Pts_In[Ø] := x1;
[9R2] 1348:   Pts_In[1] := y1;
[BR2] 1349:   Pts_In[2] := x2;
[FR2] 135Ø:   Pts_In[3] := y2;
[DT6] 1351:   VDI_call(6,Ø,Ø,2,Control,Int_In,Int_Out,
[5Y3] 1352:           Pts_In,Pts_Out,False);
[ZNØ] 1353: end;
[ØØØ] 1354:
[FL6] 1355: procedure Frame_Rect (x,y,w,h: Integer);
[MUØ] 1356: begin
[5Y2] 1357:   Pline (x,y,x+w-1,y);
[6Z3] 1358:   Pline (x+w-1,y,x+w-1,y+h-1);
[7W3] 1359:   Pline (x+w-1,y+h-1,x,y+h-1);
[6V2] 136Ø:   Pline (x,y+h-1,x,y);
[ZNØ] 1361: end;
[ØØØ] 1362:
[L26] 1363: procedure First_Rect(hand: Integer;
[TW3] 1364:           var x, y, w, h: Integer);
[MUØ] 1365: begin
[BK6] 1366:   Wind_Get(hand, WF_FIRSTXYWH, x, y, w, h);
[ZNØ] 1367: end;
[ØØØ] 1368:
[XV6] 1369: procedure Next_Rect(hand: Integer; var x,
[EM2] 137Ø:           y, w, h: Integer);
[MUØ] 1371: begin
[2C6] 1372:   Wind_Get(hand, WF_NEXTXYWH, x, y, w, h);
[ZNØ] 1373: end;
[ØØØ] 1374:
[XX5] 1375: function Min(a, b: Integer): Integer;
[MUØ] 1376: begin
[X25] 1377:   if a < b Then Min := a Else Min := b
[ZNØ] 1378: end;
[ØØØ] 1379:
[7X5] 138Ø: function Max(a, b: Integer): Integer;
[MUØ] 1381: begin
[Z35] 1382:   if a > b Then Max := a Else Max := b
[ZNØ] 1383: end;
[ØØØ] 1384:
[CC4] 1385: function Rect_Intersect
[RW4] 1386:           (x1, y1, w1, h1: Integer; var x2,
[8M4] 1387:            y2, w2, h2: Integer): Boolean;
[RNØ] 1388: var
[WW3] 1389:   tx, ty, tw, th : Integer;
[MUØ] 139Ø: begin
[AL3] 1391:   tw := min(x2 + w2, x1 + w1);
[NG3] 1392:   th := min(y2 + h2, y1 + h1);
[AA5] 1393:   tx := max(x2, x1); ty := max(y2, y1);
[NB3] 1394:   x2 := tx;          y2 := ty;
[164] 1395:   w2 := tw - tx;     h2 := th - ty;
[915] 1396:   Rect_Intersect := (tw > tx) AND
[MG1] 1397:     (th > ty);
[ZNØ] 1398: end;
[ØØØ] 1399:
[WT5] 14ØØ: procedure Set_WSize(hand, x, y, w,
[TU1] 14Ø1:                     h: Integer);
[MUØ] 14Ø2: begin
[SB6] 14Ø3:   Wind_Set(hand, WF_CurrXYWH, x, y, w, h);
[ZNØ] 14Ø4: end;
[ØØØ] 14Ø5:
[P44] 14Ø6: procedure Begin_Update;
[MUØ] 14Ø7: begin
[ØX4] 14Ø8:   int_in[Ø] := 1; (* BEG_UPDATE *)
[8A6] 14Ø9:   AES_Call(1Ø7,Int_In,Int_Out,Addr_In,
[6S1] 141Ø:           Addr_Out);
[ZNØ] 1411: end;
[ØØØ] 1412:
[WR3] 1413: procedure End_Update;
[MUØ] 1414: begin
[9X4] 1415:   int_in[Ø] := Ø; (* End_UPDATE *)
[8A6] 1416:   AES_Call(1Ø7,Int_In,Int_Out,Addr_In,
[6S1] 1417:           Addr_Out);
[ZNØ] 1418: end;
[ØØØ] 1419:
[4G6] 142Ø: procedure Border_Rect(Handle: Integer;
[TW3] 1421:           var x, y, w, h: Integer);
[MUØ] 1422: begin
[7D3] 1423:   Int_in[Ø] := handle;
[LH4] 1424:   Int_In[1] := WF_CurrXYWH;
[5A6] 1425:   AES_Call(1Ø4,Int_In,Int_Out,Addr_In,
[6S1] 1426:           Addr_Out);
[2H5] 1427:   x := Int_Out[1]; y := Int_Out[2];
[VF5] 1428:   w := Int_Out[3]; h := Int_Out[4];
[ZNØ] 1429: end;
[ØØØ] 143Ø:
[VP6] 1431: procedure Text_Heigth (Heigth: Integer);
[MUØ] 1432: begin
```

```
[UK2] 1433:   Pts_In[Ø] := Ø;
[TF3] 1434:   Pts_In[1] := Heigth;
[6S6] 1435:   VDI_call(12,Ø,Ø,1,Control,Int_In,Int_Out,
[5Y3] 1436:            Pts_In,Pts_Out,False);
[ZNØ] 1437: end;
[ØØØ] 1438:
[YP6] 1439: procedure Text_Style (style: Integer);
[MUØ] 144Ø: begin
[RD3] 1441:   Int_In[Ø] := style;
[ZA5] 1442:   VDI_call(1Ø6,Ø,1,Ø,Control,Int_In,
[9G5] 1443:            Int_Out,Pts_In,Pts_Out,False);
[ZNØ] 1444: end;
[ØØØ] 1445:
[997] 1446: procedure Text_Rotation (angle: Integer);
[MUØ] 1447: begin
[Ø53] 1448:   Int_In [Ø] := angle;
[CS6] 1449:   VDI_call(13,Ø,1,Ø,Control,Int_In,Int_Out,
[5Y3] 145Ø:            Pts_In,Pts_Out,False);
[ZNØ] 1451: end;
[ØØØ] 1452:
[MUØ] 1453: begin
[KNØ] 1454: end.
```

Listing 1. Die Unit »gembind.pas« ist die Schnittstelle zwischen Turbo Pascal und GEM

```
[KM3]    1: { Programm: alert.pas
[5F6]    2:   Funktion: Demonstriert Verwendung der
[X22]    3:             Unit gembind
[NH4]    4:   Sprache:  Turbo Pascal ab 5.Ø
[B24]    5:   Autor:    Jan Willamowius
[U65]    6:   (C) 1991 DMV Widuch GmbH & Co KG     }
[ØØØ]    7:
[FK2]    8: {$M 16384,Ø,1ØØØØ}
[TF2]    9: program Alert;
[ØØØ]   1Ø:
[H62]   11: uses Gembind;
[ØØØ]   12:
[RNØ]   13: var
[M62]   14:   btn: Integer;
[ØØØ]   15:
[MUØ]   16: begin
[R43]   17:   if Init_Gem>=Ø then
[MUØ]   18:   begin
[295]   19:     btn:=Do_Alert('[2][Ein erstes '+
[VP5]   2Ø:     'Beispiel!][ Abbruch |Weiter]',2);
[KD5]   21:     if btn=1 then btn:=Do_Alert('[3]'+
[483]   22:     '[Abbruch][ Ok ]',1)
[464]   23:     else btn:=Do_Alert('[1]'+
[973]   24:     '[Weiter][ Ok ]',1);
[ML1]   25:     Exit_Gem;
[4MØ]   26:   end
[2G4]   27:   else WriteLn ('Fehler beim '+
[6K4]   28:   'Initialisieren des GEM !');
[KNØ]   29: end.
```

Listing 2. »alert.pas« bringt eine Alert-Box auf den Bildschirm, die Benutzereingaben per Maus erwartet

```
[F74]    1: { Programm: fenster.pas
[386]    2:   Funktion: Anzeige eines GEM-Fensters
[NH4]    3:   Sprache:  Turbo Pascal ab 5.Ø
[B24]    4:   Autor:    Jan Willamowius
[R25]    5:   (C) 1991 DMV Widuch GmbH & Co KG }
[ØØØ]    6:
[9D2]    7: (*$M 16384, Ø, 1ØØØØ*)
[MX2]    8: program fenster;
[ØØØ]    9:
[H62]   1Ø: uses Gembind;
[ØØØ]   11:
[SD1]   12: var foo,
[671]   13:     handle,
[U01]   14:     event,
[NG1]   15:     x, y, w, h,
[1B5]   16:     xold, yold, hold, wold: Integer;
[XH3]   17:     msg: Message_Buffer;
[YS2]   18:     fulled: Boolean;
[RZØ]   19:     wname,
[4K3]   2Ø:     winfo: Window_Title;
[ØØØ]   21:
[HZ3]   22: procedure Draw_Window;
[MUØ]   23: begin
[7G3]   24:   Paint_Style(Solid);
[TE3]   25:   Paint_Color(White);
[LR4]   26:   Work_Rect(handle, x, y, w, h);
[NR3]   27:   Paint_Rect(x, y, w, h);
[A13]   28:   Line_Color(Black);
[U85]   29:   Pline(x+(w DIV 2),y, x, y+(h DIV 2));
[7K5]   3Ø:   Pline(x, y+(h DIV 2), x+(w DIV 2), y+h);
[KW5]   31:   Pline(x+(w DIV 2),y+h,x+w,y+(h DIV 2));
[4N5]   32:   Pline(x+w, y+(h DIV 2), x+(w DIV 2), y);
[ZNØ]   33: end;
[ØØØ]   34:
[EA7]   35: procedure Redraw(xr, yr, wr, hr : Integer);
[MUØ]   36: begin
[R22]   37:   Hide_Mouse;
[ØY4]   38:   First_Rect(handle, x, y, w, h);
[2P3]   39:   while (w <> Ø) OR (h <> Ø) DO
[MUØ]   4Ø:   begin
[HU5]   41:     if Rect_Intersect(xr, yr, wr, hr, x,
[BU1]   42:                       y, w, h) then
```

```
[MUØ]    43:     begin
[E53]    44:       Set_Clip(x, y, w, h);
[U82]    45:       Draw_Window;
[ZNØ]    46:     end;
[4R4]    47:     Next_Rect(handle, x, y, w, h);
[ZNØ]    48:   end;
[852]    49:   Show_Mouse;
[ZNØ]    5Ø: end;
[ØØØ]    51:
[MUØ]    52: begin
[493]    53:   if Init_Gem >= Ø then
[MUØ]    54:   begin
[B42]    55:     Init_Mouse;
[5G3]    56:     wname:= ' Beispiel 2 ';
[PK3]    57:     winfo:= ' Infozeile ';
[YG3]    58:     handle:= New_Window(
[HY4]    59:       G_Size OR G_Move OR G_Full OR
[AU4]    6Ø:       G_Close OR G_Name OR G_Info,
[KD2]    61:       wname, Ø, Ø, Ø, Ø);
[594]    62:     Set_Winfo(handle, winfo);
[PK5]    63:     Open_Window(handle, 3Ø, 3Ø, 2ØØ, 1ØØ);
[5K2]    64:     fulled:= false;
[P31]    65:     repeat
[S91]    66:       event:=
[4S4]    67:           Get_Event(E_Message, Ø, Ø, Ø, Ø,
[S52]    68:                              false, Ø, Ø, Ø, Ø,
[S52]    69:                              false, Ø, Ø, Ø, Ø,
[UX2]    7Ø:                              msg, foo, foo, foo,
[2D2]    71:                              foo, foo, foo);
[BB2]    72:       Begin_Update;
[E25]    73:       if event AND E_Message <> Ø then
[YE2]    74:         case msg[Ø] of
[S12]    75:           WM_Redraw :
[9P3]    76:             Redraw(msg[4], msg[5],
[BP2]    77:                    msg[6], msg[7]);
[MN1]    78:           WM_Sized,
[YY1]    79:           WM_Moved  :
[MUØ]    8Ø:             begin
[C74]    81:               Set_WSize(handle, msg[4],
[614]    82:                 msg[5], msg[6], msg[7]);
[9P3]    83:               Redraw(msg[4], msg[5],
[BP2]    84:                      msg[6], msg[7]);
[ZNØ]    85:             end;
[A12]    86:           WM_Fulled :
[MUØ]    87:             begin
[192]    88:               if fulled then
[4Ø4]    89:                 Set_WSize(handle, xold,
[DR2]    9Ø:                           yold, wold, hold)
[WP1]    91:               else begin
[5C4]    92:                 Border_Rect(handle, xold,
[YW2]    93:                             yold, wold, hold);
[1R3]    94:                 Work_Rect(Ø, x, y, w, h);
[DE3]    95:                 Set_WSize(handle, x,
[441]    96:                           y, w, h);
[ZNØ]    97:               end;
[ØD3]    98:               fulled:= NOT fulled;
[ZNØ]    99:             end;
[ZNØ]   1ØØ:         end;
[H12]   1Ø1:       End_Update;
[NP5]   1Ø2:     until (event AND E_Message <> Ø) AND
[HK3]   1Ø3:           (msg[Ø] = WM_Closed);
[TP3]   1Ø4:     Close_Window(handle);
[SU3]   1Ø5:     Delete_Window(handle);
[ML1]   1Ø6:     Exit_Gem;
[4MØ]   1Ø7:   end
[4F4]   1Ø8:   else WriteLn('Fehler beim'+
[6L4]   1Ø9:        'Initialisieren des GEM!');
[KNØ]   11Ø: end.
```

Listing 3.»fenster.pas« öffnet ein Fenster und zeichnet eine Raute. Beide reagieren auf Größenveränderungen

```
[KW3]    1: {Programm: auswahl.pas
[LY6]    2:  Funktion: Anzeige einer Dateiauswahlbox
[EK3]    3:  Sprache: Turbo Pascal
[RF4]    4:  Autor: Jürgen Willamowius
[TX4]    5:  (c)1991 DMV Widuch GmbH & Co. KG}
[ØØØ]    6:
[XX2]    7: program auswahl;
[ØØØ]    8:
[LH2]    9: {$M 16384, Ø, 1ØØØØ}
[ØØØ]   1Ø:
[H62]   11: uses gembind;
[ØØØ]   12:
[RNØ]   13: var
[AE3]   14:   Name,Pfad: path_name;
[ØØØ]   15:
[MUØ]   16: begin
[R43]   17:   if init_gem>=Ø then
[MUØ]   18:   begin
[5K5]   19:     Pfad:='C:\GEMAPPS\GEMBIND\*.PAS';
[W91]   2Ø:     Name:='';
[H74]   21:     if Get_In_File (Pfad,Name)
[SE2]   22:     then begin end;
[ML1]   23:     exit_gem;
[ZNØ]   24:   end;
[KNØ]   25: end.
```

Listing 4. Das komplexe Problem der Dateiauswahl ist in wenigen Zeilen gelöst, wie »auswahl.pas« beweist

Turbo Pascal 6.0

Laufwerke unter der Lupe

Programme, die auf unterschiedlichen PCs funktionieren sollen, müssen in der Lage sein, sich der gegebenen Hardware anzupassen. Das Betriebssystem MS-DOS verwaltet deshalb einige Tabellen, die nicht immer dokumentiert sind, aber wichtige Informationen zur Hardware enthalten.

Die entscheidende interne Tabelle von MS-DOS für Laufwerke ist der sogenannte Drive-Parameter-Block, kurz DPB. Für jedes angeschlossene Laufwerk oder Bandlaufwerk, das über einen Gerätetreiber angesprochen wird, legt MS-DOS eine entsprechende Tabelle an.

Der Aufbau eines Blocks für ein Laufwerk ist in Tabelle 1 dokumentiert. Beachten Sie, daß diese Tabelle den Aufbau des DPB unter MS-DOS 3.x wiedergibt. Unter MS-DOS 4.0 hat der Eintrag am Offset 0Fhex 2 statt 1 Byte. Deshalb kann die Version 4.0 größere Festplattenpartitionen verwalten.

Die Startadressen der nachfolgenden Einträge sind dadurch jeweils um ein Byte verschoben. Dem » Media Descriptor« (Tabelle 2) entnehmen Sie den Typ des angeschlossenen Laufwerks. Für den Zugriff auf diese Daten gibt es unter MS-DOS keinen legalen Weg. Wenn Sie undokumentierte Funktionen verwenden, stehen Ihnen drei Alternativen offen. Diese MS-DOS-Funktionen sind teilweise bereits seit der Version 1.0 vorhanden. Auch Programme wie die PC-Tools oder die Norton-Utilities bedienen sich ihrer.

Die erste Funktion, die auch das Beispielprogramm verwendet, ist die Funktion 52hex des MS-DOS-Interrupts 21hex. Nach Aufruf des Interrupts zeigen die Register »ES:BX« auf eine Tabelle, die den DOS-Info-Block, kurz DIB genannt, enthält (Tabelle 3).

Dabei zeigen die Register »ES:BX« nicht auf den Kopf der Tabelle, sondern auf den Offset 4. Dieses Feld beinhaltet die Adresse des DPB. Auf diese Weise erhalten Sie die gerade aktuellen Werte der DPBs, die teilweise nicht korrekt sind, falls seit dem letzten Systemstart kein Zugriff auf das entsprechende Laufwerk stattfand. Das Programm »readdpb.pas« (Listing) verwendet diese Funktion.

Die zweite Variante ist die Funktion 1Fhex des MS-DOS-Interrupts 21hex.

Offset	Beschreibung	Größe
0	Gerätebezeichnung des Laufwerks (0=A, 1=B, 2=C, 3=D etc.)	1 Byte
1	Hier steht die tatsächliche physikalische Nummer des Laufwerks. Angenommen, das Laufwerk A ist ein Diskettenlaufwerk für 5 1/4-Zoll-HD-Disketten mit einer Kapazität von 1,2 MByte. Mit Hilfe des Befehls »device=driver.sys /d:0 /f:0«, den Sie in die Datei »config.sys« eintragen, erzeugen Sie ein »neues Laufwerk« für 360-KByte-Diskettenformat. Wenn Sie eine Festplatte als Laufwerk C haben, erhält das zusätzliche Laufwerk nun die Gerätebezeichnung D, die nächste freie Stelle. Im DPB für das Laufwerk D beinhaltet das Feld mit dem Offset 0 nun den Wert 3 für das Gerätekennzeichen D. Physikalisch gesehen ist dieses Laufwerk aber immer noch Laufwerk A. Offset 1 des neuen DPB beinhaltet also nach wie vor den Wert 0, der das physikalische Laufwerk A bezeichnet.	1 Byte
2	Anzahl der Bytes pro Sektor. In aller Regel steht hier der Wert 512.	1 Wort
4	Interleave-Faktor. Der Interleave-Faktor zeigt an, um wieviele Sektoren die logischen gegenüber den physikalischen Sektoren verschoben sind. Die Verschiebung wird beim sogenannten Low-Level-Formatieren eines Massenspeichers durchgeführt. Dieser Wert ist nur dann korrekt gesetzt, wenn vorher ein Zugriff auf das Gerät durchgeführt wurde. Sonst enthält das Byte FEhex. Bei Disketten steht hier immer eine 1, bei Festplatten treten auch höhere Werte bis zu 6 auf. Bei modernen AT-Bus-Festplatten enthält der DPB bisweilen unsinnige Werte, ohne daß das den sicheren Betrieb stören würde.	1 Byte
5	Sektoren pro Cluster minus 1. Auch dieser Wert ist nur dann korrekt, wenn auf das Gerät bereits zugegriffen wurde. Zusätzlich müssen Sie den Wert des Feldes um 1 erhöhen. Wird zum Beispiel der Wert 3 angezeigt, so heißt das, daß vier Sektoren pro Cluster Verwendung finden, wie es bei Festplatten für ATs üblich ist.	1 Byte
6	Reservierte Sektoren für den Bootsektor	1 Wort
8	Anzahl der Dateizuordnungstabellen (englisch: File Allocation Tables, kurz FAT). Die FAT ist eine Datenstruktur auf dem Massenspeicher, die angibt, welche Cluster belegt sind und wie sie miteinander verkettet sind.	1 Byte
9	Maximale Anzahl der Hauptverzeichniseinträge	1 Wort
0Bhex	Erster Datensektor	1 Wort
0Dhex	Nummer des letzten belegten Clusters	1 Wort
0Fhex	Anzahl der Sektoren für eine FAT	1 Byte
10hex	Erster Sektor im Stammverzeichnis	1 Wort
12hex	Zeiger auf den zugehörigen Gerätetreiber. Die Verwaltung von Disketten- und Festplattenlaufwerken übernimmt in der Regel nur ein Gerätetreiber, weshalb hier dieselbe Startadresse steht. Für die interne Unterscheidung benötigt der Treiber die Angabe aus dem Byte mit Offset 1	1 Doppelwort
16hex	Media Descriptor (siehe Tabelle 2)	1 Byte
17hex	Ist der Inhalt dieses Feldes FFhex, so wurde das Gerät seit dem letzten Systemstart noch nicht angesprochen. Fehlinterpretationen anderer Werte aus dieser Tabelle sind dann nicht ausgeschlossen.	1 Byte
18hex	Zeiger auf den nächsten DPB. Ist kein weiterer Block vorhanden, steht hier der Wert FFFFhex. Auf diese Weise bilden alle DPBs eine einfach verkettete Liste.	1 Doppelwort
1Chex	Startcluster für die Suche nach freien Clustern. Wenn MS-DOS auf dem Massenspeicher nach Platz sucht, um eine Datei abzulegen, dann nimmt es die Suche bei diesem Cluster auf. Das bringt bei einer vollen Festplatte einen Zeitvorteil, weil MS-DOS nicht immer wieder Tausende von belegten Clustern in der FAT überlesen muß.	1 Wort
1Ehex	Anzahl freier Cluster auf dem Laufwerk. Wenn noch kein Zugriff auf das Laufwerk erfolgte oder der genaue Wert aus anderen Gründen unbekannt ist, steht hier FFFFhex.	1 Doppelwort

Tabelle 1. Im Disk-Parameter-Block von MS-DOS 3.x stehen wichtige Angaben zu einem Laufwerk

Wert	Beschreibung	Seiten	Spuren	Sektoren/Spur	Kapazität
F0hex	3 1/2-Zoll-Diskette	2	80	18	1,44 MByte
F8hex	Festplatte				
F9hex	5 1/4-Zoll-Diskette	2	80	15	1,2 MByte
	3 1/2-Zoll Diskette	2	80	9	720 KByte
FChex	5 1/4- Zoll-Diskette	1	40	9	180 KByte
FDhex	5 1/4- Zoll-Diskette	2	40	9	360 KByte
FEhex	5 1/4- Zoll-Diskette	1	40	8	160 KByte
FFhex	5 1/4- Zoll-Diskette	2	40	8	320 KByte

Tabelle 2. Der Media Descriptor beschreibt den Typ des angeschlossenen Massenspeichers

Nach Aufruf der Funktion 1Fhex zeigen die Register »DS:BX« auf den DPB des aktuellen Laufwerks. Zugleich führt MS-DOS bei der Benutzung dieser Funktion einen Zugriff auf das Laufwerk aus. Damit erhalten Sie stets die gerade gültigen Werte.

Seit der Version 2.0 von MS-DOS ruft diese Funktion nur mehr die Funktion 32hex mit dem Parameter 0 in »DL«

```
Auflistung der Massenspeicher mit Hilfe des DPB

Logisches Laufwerk          A       B       C       D
Physikalisches Laufwerk     A       B       C       A
Bytes pro Sektor            512     512     512     512
Interleave Faktor           254     254     7       0
Sektoren pro Cluster        1       1       4       1
Reservierte Sektoren        1       1       1       1
Anzahl der FATs             2       2       2       2
Einträge im Hauptverzeichnis 64     64      512     224
Erster belegter Sektor      9       9       167     29
Letzter belegter Cluster    352     352     17032   2372
Sektoren pro FAT            2       2       67      7
Erster Datensektor          5       5       135     15
Kopf des Gerätetreibers     0070:   0070:   0070:   1341:
                            01B6    01B6    01B6    0000
Media Descriptor            00      00      F8      F9
Gerätezugriff durchgeführt? NEIN    NEIN    JA      JA
Zeiger des nächtes DPB      02C2:   02C2:   134C:   0000:
                            8D61    8D82    0000    FFFF
Startcluster für Suche      0       0       0       0
Anzahl Freie Cluster        65535   65535   65535   2228
```

Die Ausgabe von »readdpb.pas« beweist, daß Laufwerk D über »driver.sys« realisiert ist

Offset	Bedeutung	Größe
-4	Zeiger auf den ersten MCB	1 Doppelwort
0	Zeiger auf den ersten DPB	1 Doppelwort
4	Zeiger auf den letzten MS-DOS-Puffer	1 Doppelwort
8	Zeiger auf den Uhr-Treiber	1 Doppelwort
Chex	Zeiger auf den Konsolen-Treiber	1 Doppelwort
10hex	Maximale Länge eines Sektors	1 Wort
12hex	Zeiger auf den MS-DOS-Puffer	1 Doppelwort
16hex	Zeiger auf die Pfadtabelle	1 Doppelwort
1Ahex	Zeiger auf die System File Table	1 Doppelwort
2Ehex	Einträge in der System File Table	1 Wort
30hex	Anzahl der Laufwerke	1 Byte
31hex	Letztes Laufwerk	1 Byte

Tabelle 3. Der DOS-Info-Block enthält Zeiger auf wichtige interne Datenstrukturen von MS-DOS

DOS INTERNATIONAL

DMV Daten- und Medienverlag

Der DMV-Verlag sucht zur Verstärkung des Redaktionsteams der DOS International zum nächstmöglichen Zeitpunkt

2 Redakteure

für die Bereiche Programmierung und Anwendung

Sie bringen mit:
- Kenntnisse in den wichtigsten Programmiersprachen
- sehr gute Assembler-Kenntnisse

oder
- fundierte Kenntnisse im Umgang mit den wichtigsten Standardapplikationen
- Erfahrungen mit der Benutzeroberfläche Microsoft Windows

und außerdem
- die Fähigkeit, selbständig zu arbeiten
- Flexibilität
- Engagement
- Begeisterungsfähigkeit
- Spaß am Schreiben
- und die Fähigkeit, komplexe Zusammenhänge verständlich darzustellen

Wir bieten Ihnen:
- einen anspruchsvollen Arbeitsplatz in einem jungen, engagierten Team
- leistungsgerechte Bezahlung
- die üblichen Sozialleistungen
- das attraktive Freizeitangebot des Großraums München
- sowie die übliche Menge Streß, Spaß, Hektik und Arbeit einer Zeitschriftenredaktion

Sind Sie interessiert?

Dann senden Sie Ihre Bewerbung mit den üblichen Unterlagen an das

DMV-Redaktionsbüro München, »Stellenanzeige«, Gruberstr. 46c, 8011 Poing bei München

Für eine erste unverbindliche Kontaktaufnahme steht Ihnen Herr Ulrich Eike unter der Telefonnummer 08121/769-20 als Ansprechpartner zur Verfügung.

auf. Verwenden Sie also besser gleich die im folgenden beschriebene neue Funktion.

Die dritte Alternative bedient sich der Funktion 32hex des Interrupts 21hex. Bei dieser Variante übergeben Sie dem DL-Register die Gerätenummer des gewünschten Gerätes. 0 liefert den DPB des aktuellen Laufwerks, 1 den von Laufwerk A, 2 den von B etc. Nach Aufruf der Funktion, die ihrerseits das gewählte Laufwerk kurz anspricht, zeigen die Register »DS:BX« auf den entsprechenden Disk-Parameter-Block. Falls das angegebene Laufwerk nicht existiert, liefert die Funktion im Register »AL« den Wert FFhex zurück.

Das Programm »readdpb.pas« liest der Reihe nach sämtliche DPBs aus und zeigt sie am Bildschirm an. Im Bild sehen Sie die Ausgabe des Programms auf einem Computer mit zwei Diskettenlaufwerken, einer Festplatte und einem vierten, mit »driver.sys« eingerichteten Laufwerk. Entsprechend weist auch der Gerätetreiber eine andere Startadresse auf.

Den Aufbau der Bildschirmmaske demonstriert das Listing ab Zeile 124. Die hierbei verwendeteten Grafikzeichen geben Sie beispielsweise über [Alt 201] ein.

(Werner Jakob/wn)

```
[4S3]    1: {Programm: readdbp.pas
[8D6]    2:  Funktion: Auslesen der Drive Parameter
[MB1]    3:            Blöcke
[NH4]    4:  Sprache:  Turbo Pascal ab 5.Ø
[SG3]    5:  Autor:    Werner Jakob
[TX4]    6:  (c)1991 DMV Widuch GmbH & Co. KG}
[ØØØ]    7:
[S62]    8: uses Crt,Dos;
[ØØØ]    9:
[ØØØ]   1Ø:
[STØ]   11: type
[ØT3]   12:   { Zeiger auf den DPB }
[BL3]   13:   DPB3Zeiger=^DPB3Feld;
[DL3]   14:   DPB4Zeiger=^DPB4Feld;
[ØØØ]   15:
[SS6]   16:   {Struktur zur Aufnahme der DPB-Felder bei
[D12]   17:   MS-DOS 3.x}
[WF1]   18:   DPB3Feld=
[Y11]   19:   record
[725]   2Ø:     LL:Byte;{Logisches Laufwerk}
[866]   21:     LP:Byte;{Physikalisches Laufwerk}
[BR4]   22:     BS:Word;{Bytes pro Sektor}
[B85]   23:     IL:Byte;{Interleave-Faktor}
[XH5]   24:     SC:Byte;{Sektoren pro Cluster}
[HS5]   25:     RS:Word;{Reservierte Sektoren}
[L24]   26:     AF:Byte;{Anzahl FAT's}
[7T3]   27:     AH:Word;{Einträge im
[W63]   28:     Hauptverzeichnis}
[SC4]   29:     ES:Word;{Erster belegter
[XL1]   3Ø:     Sektor}
[NE4]   31:     LC:Word;{Letzter belegter
[DK1]   32:     Cluster}
[LS4]   33:     SF:Byte;{Sektoren pro FAT}
[175]   34:     ED:Word;{Erster Datensektor}
[U35]   35:     ZGT:pointer;{Zeiger auf den
[XL1]   36:     Anfang des
[LK2]   37:     entsprechenden
[F43]   38:     Gerätetreibers}
[SR4]   39:     MD:Byte;{Media Descriptor}
[U64]   4Ø:     UF:Byte;{Anzeige Byte ob
[9M3]   41:     auf das Gerät bereits
[XU3]   42:     ein Zugriff erfolgte}
[ØK5]   43:     NDPB:pointer;{Adresse nächster
[BSØ]   44:     DBP}
[6X3]   45:     NFC:Word;{Suchstart}
[RC4]   46:     FC:Word;{Freie Cluster}
[ZNØ]   47:   end;
[ØØØ]   48:
[SS6]   49:   {Struktur zur Aufnahme der DPB-Felder bei
[E12]   5Ø:   MS-DOS 4.x}
[XF1]   51:   DPB4Feld=
[Y11]   52:   record
[725]   53:     LL:Byte;{Logisches Laufwerk}
[866]   54:     LP:Byte;{Physikalisches Laufwerk}
[BR4]   55:     BS:Word;{Bytes pro Sektor}
[B85]   56:     IL:Byte;{Interleave-Faktor}
[XH5]   57:     SC:Byte;{Sektoren pro Cluster}
[HS5]   58:     RS:Word;{Reservierte Sektoren}
[L24]   59:     AF:Byte;{Anzahl FAT's}
[7T3]   6Ø:     AH:Word;{Einträge im
[W63]   61:     Hauptverzeichnis}
[SC4]   62:     ES:Word;{Erster belegter
[XL1]   63:     Sektor}
[NE4]   64:     LC:Word;{Letzter belegter
[DK1]   65:     Cluster}
[AS4]   66:     SF:Word;{Sektoren pro FAT}
[175]   67:     ED:Word;{Erster Datensektor}
[U35]   68:     ZGT:pointer;{Zeiger auf den
[XL1]   69:     Anfang des
[LK2]   7Ø:     entsprechenden
[F43]   71:     Gerätetreibers}
[SR4]   72:     MD:Byte;{Media Descriptor}
[U64]   73:     UF:Byte;{Anzeige Byte ob
[9M3]   74:     auf das Gerät bereits
[XU3]   75:     ein Zugriff erfolgte}
[ØK5]   76:     NDPB:pointer;{Adresse nächster
[BSØ]   77:     DBP}
[6X3]   78:     NFC:Word;{Suchstart}
[RC4]   79:     FC:Word;{Freie Cluster}
[ZNØ]   8Ø:   end;
[ØØØ]   81:
[8P4]   82:   {Aufnahme von Hex-Zeichen}
```

```
[4D3]   83:   hexword=string[2];
[ØØØ]   84:
[RNØ]   85: var
[VY2]   86:   DosVersion:Byte;
[GU2]   87:   regs:Registers;
[C44]   88:   Feld3Zeiger:DPB3Zeiger;
[F44]   89:   Feld4Zeiger:DPB4Zeiger;
[ØØØ]   9Ø:
[Y85]   91:   {Zeiger auf die zweite Adresse
[LD1]   92:    im DIP}
[PG3]   93:   DIBZeiger:^pointer;
[ØØØ]   94:
[2E5]   95:   linie1,linie2,linie3:string[8Ø];
[ØØØ]   96:
[S97]   97: {Die Funktion ermittelt nur die Hauptver-
[ZØ5]   98: sionsnummer, die in AL steht}
[B35]   99: function getdosversion:Byte;
[MUØ]  1ØØ: begin
[LX1]  1Ø1:   regs.AH:=$3Ø;
[3Ø2]  1Ø2:   MsDos(regs);
[G74]  1Ø3:   getdosversion:=regs.al;
[ZNØ]  1Ø4: end;
[ØØØ]  1Ø5:
[XF5]  1Ø6: {Diese Funktion formt ein Byte in
[X76]  1Ø7: zwei entsprechende ASCII-Zeichen um}
[MC6]  1Ø8: function machhex(zahl1:Byte):hexword;
[TX5]  1Ø9: const hexwerte:array[Ø..15]of Char=
[G43]  11Ø: ('Ø','1','2','3','4','5','6',
[G53]  111:       '7','8','9','A','B','C','D',
[Ø21]  112:       'E','F');
[ØØØ]  113:
[PD2]  114:       var b1,b2:Byte;
[MF2]  115:       ergeb:hexword;
[MUØ]  116: begin
[SG2]  117:   b1:=zahl1 AND $ØF;
[7C2]  118:   b2:=zahl1 SHR 4;
[E64]  119:   ergeb[1]:=hexwerte[b2];
[G64]  12Ø:   ergeb[2]:=hexwerte[b1];
[YN2]  121:   machhex:=ergeb;
[ZNØ]  122: end;
[ØØØ]  123:
[G45]  124: {Aufbau der Bildschirmmaske}
[EL4]  125: procedure Bildschirmmaske;
[PD2]  126: var I:Integer;
[MUØ]  127: begin
[K5E]  128:   linie1:='╔═══════════════════════════════════════'+
[SVC]  129:   '═══════╦═══════╦═══════╦═══════╦═══════╗'+
[1CE]  13Ø:   '╠═══════╬═══════╬═══════╬═══════╬═══════╣';
[P94]  131:   linie2:='║                                       '+
[FY4]  132:   '       ║       ║       ║       ║       ║'+
[344]  133:   '       ╠═══════╬═══════╬═══════╬═══════╣';
[N5E]  134:   linie3:='╚═══════════════════════════════════════'+
[GVC]  135:   '═══════╩═══════╩═══════╩═══════╩═══════╝'+
[RBE]  136:   '═══════╩═══════╩═══════╩═══════╩═══════╝';
[VZ1]  137:   GotoXY(15,2);
[655]  138:   Write('Auflistung der Massen');
[DR5]  139:   Write('speicher mit Hilfe des DPB');
[RV1]  14Ø:   GotoXY(2,3);
[PA2]  141:   Write(linie1);
[ØK2]  142:   for I:=4 to 23 do
[MUØ]  143:   begin
[TX1]  144:     GotoXY(2,I);
[RA2]  145:     Write(linie2);
[ZNØ]  146:   end;
[4Ø2]  147:   GotoXY(2,24);
[SA2]  148:   Write(linie3);
[M73]  149:   TextColor(yellow);
[VV1]  15Ø:   GotoXY(3,4);
[LG4]  151:   Write('Logisches Laufwerk');
[YV1]  152:   GotoXY(3,5);
[MH5]  153:   Write('Physikalisches Laufwerk');
[1W1]  154:   GotoXY(3,6);
[RB4]  155:   Write('Bytes pro Sektor');
[4W1]  156:   GotoXY(3,7);
[UE4]  157:   Write('Interleave Faktor');
[7W1]  158:   GotoXY(3,8);
[E65]  159:   Write('Sektoren pro Cluster');
[AW1]  16Ø:   GotoXY(3,9);
[BØ5]  161:   Write('Reservierte Sektoren');
[KZ1]  162:   GotoXY(3,1Ø);
[RV3]  163:   Write('Anzahl der FATs');
```

```
[PZ1]  164:    GotoXY(3,11);
[VK6]  165:    Write('Einträge im Hauptverzeichnis');
[UZ1]  166:    GotoXY(3,12);
[E55]  167:    Write('Erster belegter Sektor');
[YZ1]  168:    GotoXY(3,13);
[CT5]  169:    Write('Letzter belegter Cluster');
[2Ø2]  17Ø:    GotoXY(3,14);
[XC4]  171:    Write('Sektoren pro FAT');
[6Ø2]  172:    GotoXY(3,15);
[AM4]  173:    Write('Erster Datensektor');
[AØ2]  174:    GotoXY(3,16);
[4L5]  175:    Write('Kopf des Gerätetreibers');
[KØ2]  176:    GotoXY(3,18);
[C94]  177:    Write('Media Descriptor');
[PØ2]  178:    GotoXY(3,19);
[UR6]  179:    Write('Gerätezugriff durchgeführt?');
[NZ1]  18Ø:    GotoXY(3,2Ø);
[G15]  181:    Write('Zeiger des nächtes DPB');
[XZ1]  182:    GotoXY(3,22);
[XM5]  183:    Write('Startcluster für Suche');
[1Ø2]  184:    GotoXY(3,23);
[6Z4]  185:    Write('Anzahl Freie Cluster');
[ZNØ]  186:  end;
[ØØØ]  187:
[ØØØ]  188:
[YT5]  189:  {Ermitteln und auswerten des DPB}
[U24]  19Ø:  procedure AbfrageINT52H;
[4E3]  191:  var spalte:Integer;
[RR2]  192:  weiter:Boolean;
[SZ1]  193:  byte1:Byte;
[MUØ]  194:  begin
[7A2]  195:    weiter:=True;
[AV1]  196:    spalte:=32;
[7U5]  197:    {Nach Aufruf des Interruptes 52hex
[UK4]  198:    zeigt des Register ES auf die
[TN3]  199:    Segmentadresse und das
[VR5]  2ØØ:    Register BX auf die Offsetadresse
[8E4]  2Ø1:    des Disk Parameterblocks}
[XX1]  2Ø2:    regs.AH:=$52;
[3Ø2]  2Ø3:    MsDos(regs);
[LA5]  2Ø4:    DIBZeiger:=Ptr(regs.ES,regs.bx);
[FF3]  2Ø5:    TextColor(lightred);
[DC3]  2Ø6:    if DosVersion=3 then
[MUØ]  2Ø7:    begin
[CA4]  2Ø8:      Feld3Zeiger:=DIBZeiger^;
[F96]  2Ø9:      {Die Parameter werden nun ausgewertet
[HC6]  21Ø:      bis der Offset-Zeiger auf den nächsten
[Y13]  211:      DPB FFFFhex ist.}
[P31]  212:      repeat
[CT2]  213:        GotoXY(spalte,4);
[R55]  214:        Write(Char(Feld3Zeiger^.LL+65));
[GT2]  215:        GotoXY(spalte,5);
[365]  216:        Write(Char(Feld3Zeiger^.LP+65));
[MT2]  217:        GotoXY(spalte,6);
[GW3]  218:        Write(Feld3Zeiger^.BS);
[ST2]  219:        GotoXY(spalte,7);
[9W3]  22Ø:        Write(Feld3Zeiger^.IL);
[WT2]  221:        GotoXY(spalte,8);
[YZ3]  222:        Write(Feld3Zeiger^.SC+1);
[ØU2]  223:        GotoXY(spalte,9);
[ØX3]  224:        Write(Feld3Zeiger^.RS);
[MX2]  225:        GotoXY(spalte,1Ø);
[NV3]  226:        Write(Feld3Zeiger^.AF);
[NX2]  227:        GotoXY(spalte,11);
[TV3]  228:        Write(Feld3Zeiger^.AH);
[PX2]  229:        GotoXY(spalte,12);
[LW3]  23Ø:        Write(Feld3Zeiger^.ES);
[RX2]  231:        GotoXY(spalte,13);
[UV3]  232:        Write(Feld3Zeiger^.LC);
[SX2]  233:        GotoXY(spalte,14);
[7W3]  234:        Write(Feld3Zeiger^.SF);
[TX2]  235:        GotoXY(spalte,15);
[NV3]  236:        Write(Feld3Zeiger^.ED);
[UX2]  237:        GotoXY(spalte,16);
[D26]  238:        byte1:=Seg(Feld3Zeiger^.ZGT^)SHR 8;
[6K3]  239:        Write(machhex(byte1));
[EØ5]  24Ø:        byte1:=Seg(Feld3Zeiger^.ZGT^)
[RD1]  241:        AND $ØØFF;
[214]  242:        Write(machhex(byte1),':');
[VX2]  243:        GotoXY(spalte,17);
[F36]  244:        byte1:=Ofs(Feld3Zeiger^.ZGT^)SHR 8;
[6K3]  245:        Write(machhex(byte1));
[G15]  246:        byte1:=Ofs(Feld3Zeiger^.ZGT^)
[RD1]  247:        AND $ØØFF;
[6K3]  248:        Write(machhex(byte1));
[WX2]  249:        GotoXY(spalte,18);
[G65]  25Ø:        Write(machhex(Feld3Zeiger^.MD));
[XX2]  251:        GotoXY(spalte,19);
[C64]  252:        if Feld3Zeiger^.UF=Ø then
[9C2]  253:        Write('JA')else
[F62]  254:        Write('NEIN');
[SX2]  255:        GotoXY(spalte,2Ø);
[T26]  256:        byte1:=Seg(Feld3Zeiger^.NDPB^)SHR 8;
[6K3]  257:        Write(machhex(byte1));
[655]  258:        byte1:=Seg(Feld3Zeiger^.NDPB^)
[RD1]  259:        AND $ØØFF;
[214]  26Ø:        Write(machhex(byte1),':');
[TX2]  261:        GotoXY(spalte,21);
[V36]  262:        byte1:=Ofs(Feld3Zeiger^.NDPB^)SHR 8;
[6K3]  263:        Write(machhex(byte1));
```

```
[865]  264:        byte1:=Ofs(Feld3Zeiger^.NDPB^)
[RD1]  265:        AND $ØØFF;
[6K3]  266:        Write(machhex(byte1));
[UX2]  267:        GotoXY(spalte,22);
[3Y3]  268:        Write(Feld3Zeiger^.NFC);
[VX2]  269:        GotoXY(spalte,23);
[MV3]  27Ø:        Write(Feld3Zeiger^.FC);
[RT2]  271:        spalte:=spalte+6;
[SA5]  272:        if Ofs(Feld3Zeiger^.NDPB^)=$FFFF
[UB3]  273:        then weiter:=false;
[UE5]  274:        Feld3Zeiger:=Feld3Zeiger^.NDPB;
[TG5]  275:      until(weiter=false)OR(spalte>74);
[4MØ]  276:    end
[ZRØ]  277:    else
[MUØ]  278:    begin
[EA4]  279:      Feld4Zeiger:=DIBZeiger^;
[F96]  28Ø:      {Die Parameter werden nun ausgewertet
[HC6]  281:      bis der Offset-Zeiger auf den nächsten
[Y13]  282:      DPB FFFFhex ist.}
[P31]  283:      repeat
[CT2]  284:        GotoXY(spalte,4);
[S55]  285:        Write(Char(Feld4Zeiger^.LL+65));
[GT2]  286:        GotoXY(spalte,5);
[465]  287:        Write(Char(Feld4Zeiger^.LP+65));
[MT2]  288:        GotoXY(spalte,6);
[MW3]  289:        Write(Feld4Zeiger^.BS);
[ST2]  29Ø:        GotoXY(spalte,7);
[DW3]  291:        Write(Feld4Zeiger^.IL);
[WT2]  292:        GotoXY(spalte,8);
[2Ø4]  293:        Write(Feld4Zeiger^.SC+1);
[ØU2]  294:        GotoXY(spalte,9);
[4X3]  295:        Write(Feld4Zeiger^.RS);
[MX2]  296:        GotoXY(spalte,1Ø);
[TV3]  297:        Write(Feld4Zeiger^.AF);
[NX2]  298:        GotoXY(spalte,11);
[XV3]  299:        Write(Feld4Zeiger^.AH);
[PX2]  3ØØ:        GotoXY(spalte,12);
[RW3]  3Ø1:        Write(Feld4Zeiger^.ES);
[RX2]  3Ø2:        GotoXY(spalte,13);
[YV3]  3Ø3:        Write(Feld4Zeiger^.LC);
[SX2]  3Ø4:        GotoXY(spalte,14);
[BW3]  3Ø5:        Write(Feld4Zeiger^.SF);
[TX2]  3Ø6:        GotoXY(spalte,15);
[TV3]  3Ø7:        Write(Feld4Zeiger^.ED);
[UX2]  3Ø8:        GotoXY(spalte,16);
[E26]  3Ø9:        byte1:=Seg(Feld4Zeiger^.ZGT^)SHR 8;
[6K3]  31Ø:        Write(machhex(byte1));
[FØ5]  311:        byte1:=Seg(Feld4Zeiger^.ZGT^)
[RD1]  312:        AND $ØØFF;
[214]  313:        Write(machhex(byte1),':');
[VX2]  314:        GotoXY(spalte,17);
[G36]  315:        byte1:=Ofs(Feld4Zeiger^.ZGT^)SHR 8;
[6K3]  316:        Write(machhex(byte1));
[H15]  317:        byte1:=Ofs(Feld4Zeiger^.ZGT^)
[RD1]  318:        AND $ØØFF;
[6K3]  319:        Write(machhex(byte1));
[WX2]  32Ø:        GotoXY(spalte,18);
[M65]  321:        Write(machhex(Feld4Zeiger^.MD));
[XX2]  322:        GotoXY(spalte,19);
[D64]  323:        if Feld4Zeiger^.UF=Ø then
[9C2]  324:        Write('JA')else
[F62]  325:        Write('NEIN');
[SX2]  326:        GotoXY(spalte,2Ø);
[U26]  327:        byte1:=Seg(Feld4Zeiger^.NDPB^)SHR 8;
[6K3]  328:        Write(machhex(byte1));
[755]  329:        byte1:=Seg(Feld4Zeiger^.NDPB^)
[RD1]  33Ø:        AND $ØØFF;
[214]  331:        Write(machhex(byte1),':');
[TX2]  332:        GotoXY(spalte,21);
[W36]  333:        byte1:=Ofs(Feld4Zeiger^.NDPB^)SHR 8;
[6K3]  334:        Write(machhex(byte1));
[965]  335:        byte1:=Ofs(Feld4Zeiger^.NDPB^)
[RD1]  336:        AND $ØØFF;
[6K3]  337:        Write(machhex(byte1));
[UX2]  338:        GotoXY(spalte,22);
[7Y3]  339:        Write(Feld4Zeiger^.NFC);
[VX2]  34Ø:        GotoXY(spalte,23);
[SV3]  341:        Write(Feld4Zeiger^.FC);
[RT2]  342:        spalte:=spalte+6;
[TA5]  343:        if Ofs(Feld4Zeiger^.NDPB^)=$FFFF
[UB3]  344:        then weiter:=false;
[ZE5]  345:        Feld4Zeiger:=Feld4Zeiger^.NDPB;
[TG5]  346:      until(weiter=false)OR(spalte>74);
[ZNØ]  347:    end;
[ZNØ]  348:  end;
[ØØØ]  349:
[MUØ]  35Ø:  begin
[VU4]  351:    DosVersion:=getdosversion;
[YØ3]  352:    TextColor(white);
[KL3]  353:    TextBackGround(blue);
[DB1]  354:    ClrScr;
[1W2]  355:    Bildschirmmaske;
[5A2]  356:    AbfrageINT52H;
[P31]  357:    repeat
[K13]  358:    until KeyPressed;
[DB1]  359:    ClrScr;
[KNØ]  36Ø:  end.
```

»readdpb.pas« liest sämtliche Disk-Parameter-Blöcke aus und zeigt sie auf dem Bildschirm an

Interruptcontroller und serielle Schnittstelle

Controller fest im Griff

Rechnernetze und digitaler Nachrichtenaustausch gehören mittlerweile zum Alltag eines jeden PC-Anwenders. Wie man die durch den Rechner gegebene Hardware zur Datenübertragung nutzt, ist jedoch weitgehend unbekannt. Höchste Zeit also, den Aufbau des Interruptcontrollers und der seriellen Schnittstelle zu beschreiben. Als kleines Bonbon gibt es eine Funktionsbibliothek zu deren Programmierung.

Die direkte Programmierung der Hardwarebauteile ist eines der stiefmütterlich behandelten Themengebiete des PC. Das liegt zweifellos daran, daß es sich um ein sehr komplexes und trockenes Thema handelt. Dabei sind Sie erst durch die genaue Kenntnis der Hardware in der Lage, die Leistungsstärke Ihres Rechners auch wirklich auszureizen.

Der Interruptcontroller 8259 ist einer der wichtigsten und wirkungsvollsten Bausteine des Rechners. Er ist mitverantwortlich für die gesamte Kommunikation zwischen CPU und Außenwelt. Der Baustein zur asynchronen Datenübertragung (Asynchronous Communication Element, kurz ACE) 8250 arbeitet im vorgestellten Beispiel mit dem Interruptcontroller zusammen, um Daten zwischen zwei Rechnern auszutauschen. Durch Kenntnisse über den Aufbau und die Struktur dieser Bausteine programmieren Sie die sogenannten Kommunikationsports in einer beliebigen Hochsprache.

Daß die direkte Programmierung nicht nur Assembler-Programmierern vorbehalten ist, verdeutlicht die in C geschriebene Funktionsbibliothek. Diese Bibliothek läuft sowohl unter Turbo C als auch unter Quick C. Um Unterschiede zwischen den beiden Dialekten abzufangen, verwenden Sie »compatib.c« (Listing 1). Zunächst folgen eingehende Erklärungen über den Interruptcontroller 8259. Auf diese Grundlagen aufbauend, geht es an die Programmierung des Prozessors 8250. Schließlich lernen Sie das 7-Schichten-Modell kennen, das zeigt, welche Ebenen eine sichere Datenübertragung durchlaufen muß.

Nahezu jedes Rechnersystem arbeitet mit einer asynchronen Interruptverarbeitung. Damit ist allerdings nicht das Aufrufen von Software-Interrupts, beispielsweise des MS-DOS-Interrupts 21hex, gemeint. Die Sofware-Interrupts sind lediglich eine Schnittstelle für Systementwickler, um auf die Funktionen von MS-DOS oder des BIOS zuzugreifen. Die Hardware-Interrupts hingegen erlauben die Verarbeitung asynchroner Ereignisse, die von der peripheren Hardware des Rechners ausgelöst werden. Damit kann der Rechner auf Ereignisse reagieren, ohne auf diese warten zu müssen.

Ein Beispiel dafür ist der Tastaturinterrupt. Wenn Sie eine Taste drücken, meldet der Tastaturprozessor dieses Ereignis dem Interruptcontroller 8259. Daraufhin generiert der Interruptcontroller den Tastaturinterrupt 9, der das laufende Programm unterbricht und die CPU veranlaßt, die Tastatur-Interruptroutine abzuarbeiten. Nach deren Beendigung fährt die CPU mit dem unterbrochenen Programm fort. Da die Ereignisse asynchron zu einer Programmausführung auftreten können, muß auch ihre Verarbeitung asynchron ablaufen.

Notwendig ist die asynchrone Bearbeitung von Ereignissen vor allem bei der Kommunikation mit der externen Hardware, wie beispielsweise Drucker, Tastatur und Festplatte. Diese Ereignisse laufen in Echtzeit ab und werden nicht direkt von der CPU kontrolliert. Wenn Sie beispielsweise eine Taste drücken und der Tastaturprozessor übernimmt den Tastencode nicht sofort, überschreibt ein erneuter Tastendruck den alten Tastencode, der damit verloren geht. Die notwendige sofortige Reaktion ist nur durch ein Interruptsystem zu gewährleisten.

Eine Alternative besteht im Pollingbetrieb der Schnittstellen. Dazu muß die CPU die einzelnen Geräte permanent auf ihre Aktivitäten abfragen und entsprechend reagieren. Bei dieser Methode dürfen zwei Ereignisse keinesfalls gleichzeitig auftreten, da sonst mindestens eines verloren geht. Ganz abgesehen davon kann die CPU kein Programm mehr verarbeiten, da sie die gesamte Zeit in der Abfrageroutine für Ereignisse verbringt.

Für die gesamte Interruptverarbeitung ist beim PC der 8259 verantwortlich – wegen seiner Aufgabe Interruptprozessor oder Interruptcontroller genannt. Hinter diesem Baustein verbergen sich die im folgenden aufgezählten Funktionen, die weit über das sofortige Reagieren auf asychrone Ereignisse hinausgehen:

- Der Interruptcontroller kann einen Interrupt auslösen, unterbinden oder erlauben.
- Falls mehrere Ereignisse gleichzeitig auftreten, entscheidet der Controller, welches Ereignis zuerst bearbeitet wird und legt die Prioritäten fest.
- Er verknüpft die Vielzahl der Interruptleitungen in logischer Weise mit der einen Interruptleitung der CPU.

Diese Leistungmerkmale des Interruptcontrollers 8259 ergeben in Verbindung mit der CPU ein Team, dessen Einsatz Sie in Programmen nicht hoch genug einschätzen können. Die Interruptprozeduren sind so flexibel einsetzbar, daß sie nahezu in allen professionellen Programmen vorkommen (beispielsweise Tastaturtreibern, Anwendungen, die über Aktivierungstasten (Hotkeys) aufgerufen werden, Uhren und so weiter).

Jedes externe Gerät und jede Schnittstelle, die einen Hardware-Interrupt

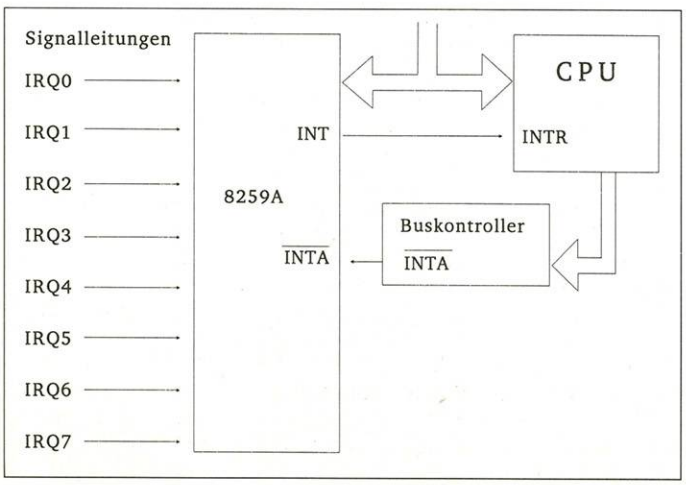


Bild 1. Der Interruptcontroller 8259

auslösen, besitzen eine Verbindung zum Interruptcontroller 8259. Falls bei einem dieser Geräte ein Ereignis eintritt, schickt es über diese Verbindung (Interruptleitung) ein Signal. Der Interruptcontroller verknüpft das ankommende Signal mit den Signalen, die noch von anderen Geräten an ihn gemeldet wurden. Falls kein Signal höherer Priorität vorliegt, leitet er den Interrupt weiter an die CPU und übergibt die Interruptnummer. Mit dieser Nummer springt die CPU indirekt zur entsprechenden Interruptprozedur und verarbeitet das Ereignis (Bild 1).

In jedem PC oder XT finden Sie genau einen Interruptcontroller 8259. Die Signalleitungen sind dort mit den Ereignissen auf die in Tabelle 1 gezeigte Weise gekoppelt. Im Unterschied dazu sind im AT zwei Interruptcontroller vorhanden, da der AT mehr Ereignisse unterstützt. Die Belegung der Signalleitungen für den AT finden Sie in Tabelle 2.

Wie schon erwähnt, besitzt die CPU eigentlich nur eine einzige Signalleitung, über die ein externes Gerät einen Interrupt auslöst. Da beim PC nur ein Interruptcontroller existiert, ist dessen INT-Ausgang direkt mit der Signalleitung der CPU verbunden.

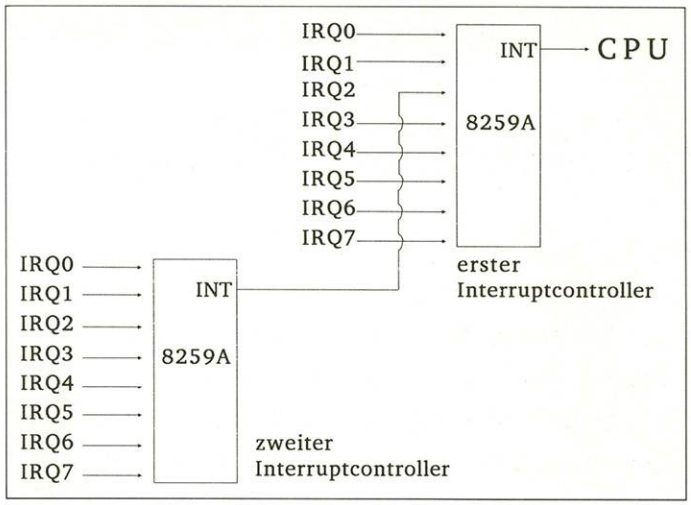


Bild 2. Die Kaskadierung der zwei Interruptcontroller beim AT

Alle Interruptanforderungen laufen über den Controller 8259

Beim AT, der zwei Interruptcontroller besitzt, ist nur der erste direkt mit der CPU verbunden, während der zweite eine Signalleitung des ersten Controllers belegt (Bild 2). Man nennt dieses Verfahren »Kaskadierung der Interruptcontroller«. Tritt ein Ereignis auf, für das der zweite Interruptcontroller zuständig ist, schickt er sein Interruptsignal an den ersten Controller, der das Signal wiederum an die CPU schickt.

Damit Sie sich den Ablauf einer Interruptbearbeitung besser vorstellen können, ein Beispiel: das Drücken einer Taste. Dabei wird davon ausgegangen, daß der Interrupt zugelassen ist und kein Interrupt höherer Priorität zur Bearbeitung anliegt.

Mit dem Betätigen einer Taste durch den Anwender registriert der Tastaturprozessor ein Ereignis, für das er verantwortlich ist. Auf welche Weise die Tastencodes eingelesen und an ein Anwendungsprogramm übergeben werden, soll an dieser Stelle nicht beschrieben werden. Der Tastaturprozessor schickt sofort ein Signal an den Interruptcontroller 8259. Dieser stellt den Interrupt IRQ 1 fest und gibt das Signal über die INT-Leitung an die CPU weiter. Falls im Statusregister der CPU das Interrupt-Flag gesetzt (das heißt, die Annahme von Interrupts zugelassen) ist, nimmt die CPU Verbindung mit dem Interruptcontroller auf. Nachdem die CPU den Bus über den Buscontroller belegt hat, übergibt der Interruptprozessor die entsprechende Interruptvektor-Nummer (in diesem Fall 9) an die CPU. Die CPU multipliziert diese Adresse mit vier und ermittelt so die absolute Adresse, über die sie indirekt zur eigentlichen Interruptprozedur springt. Normalerweise arbeitet die CPU beim Eintritt eines Hardware-Interrupts ein Anwendungsprogramm ab, das von der Interruptverarbeitung nicht beeinflußt werden darf. Daher sichert die CPU zu Beginn jeder Interruptprozedur die aktuellen Inhalte ihrer Register.

Beim Beispiel des Tastendrucks liest die Interruptprozedur nun den Tastencode vom Tastaturprozessor ein und stellt ihn im Tastaturpuffer bereit. Mit dem Verlassen der Interruptprozedur wird der Zustand der CPU wieder hergestellt, den sie beim Eintreten in die Interruptprozedur hatte.

Der Interruptcontroller 8259 besitzt zur Verarbeitung der auftretenden Ereignisse insgesamt drei Register. Jedem Ereignis ist je ein Bit zugeordnet.

- Das Interrupt-Mask-Register (IMR) legt fest, ob beim Empfangen eines Signals der entsprechende Interrupt an die CPU weitergegeben oder das Signal ignoriert wird. So verbieten oder erlauben Sie bestimmte Interrupts.
- Das Interrupt-Request-Register (IRR) speichert die ankommenden Signale der Signalleitungen.
- Das Interrupt-Service-Register (ISR) zeigt an, welche Interruptereignisse gerade in Bearbeitung sind.

Außer diesen drei Registern enthält der 8259 die Steuerungslogik sowie verschiedene Puffer (Bild 3). Auch für die Prioritäten der Abarbeitung existiert eine Logik, so daß der Interrupt mit der höchsten Priorität zuerst abgearbeitet wird und erst danach die anderen. Im sogenannten Kaskadenpuffer wird die Nummer des Interruptcontrollers gespeichert, bei dem ein Ereignis eintrat. Wie bereits erwähnt, ist es durchaus denkbar, daß zwei völlig unabhängige Ereignisse gleichzeitig eintreten. Diese Kollision von Ereignissen löst der Interruptcontroller auf. Durch eine Prioritätsverteilung ist festgelegt, in welcher Reihenfolge die einlaufenden Interruptanforderungen bearbeitet werden. Dieses Prioritätenschema läßt auch während der Bearbeitung eines Interrupts den Aufruf und die Bearbeitung von Interrupts mit höherer Priorität zu. Ohne Prioritätenschema wäre dies nicht realisierbar.

Auch gleichzeitig eintretende Ereignisse sind kein Problem

Nach der Beendigung einer Interruptprozedur wäre nicht festgelegt, welcher der übrigen Interrupts als nächster zu bearbeiten ist. Beim Prioritätenschema hingegen ist nur der in der Rangfolge nächsttiefere Interrupt im »ISR« zu suchen und mit dessen Abarbeitung fortzufahren. Dieses Verfahren nennt man den »vollverschachtelten Modus« der Interruptverarbeitung. Es ist einer der Arbeitsmodi des 8259.

Die Prioritäten der Interrupts sind von 0 bis 7 den Bits der 3 Interruptregister zugeordnet (jedes der drei Register enthält alle Interrupts). Im Normalfall hat Bit 0 die höchste Priorität und so weiter. Um den vollverschachtelten Modus besser zu verstehen, schauen Sie sich Bild 4 an.

Bit	Belegung
0	Timer
1	Tastatur
2	frei
3	zweite serielle Schnittstelle
4	erste serielle Schnittstelle
5	Festplatte beim XT
6	Diskettenlaufwerk
7	paralleler Drucker

Tabelle 1. Die Belegung der Signalleitungen des 8259 beim PC

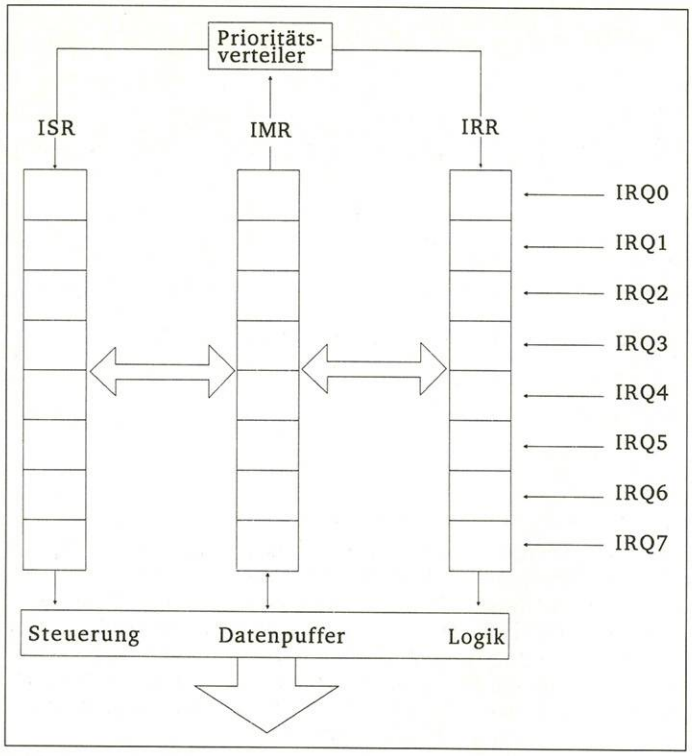


Bild 3. Die internen Register des 8259

Zunächst tritt ein Interrupt durch Priorität 6 auf. Damit sind nur noch die Interrupts mit höherer Priorität zugelassen. In der nächsten Sequenz wird nun der Interrupt 3 angefordert. Dadurch wird die Bearbeitung von Interrupt 6 unterbrochen und Interrupt 3 abgearbeitet. Nach Beendigung dieser Interruptprozedur fährt der Controller mit Interrupt 6 fort.

Die Kollision mehrerer Interrupts kann auch zu einem Datenverlust führen. Dauert die Bearbeitung eines Interrupts zu lange, können die Daten einer zweiten Interruptanforderung von einer dritten überschrieben werden. Beispielsweise darf bei einer Übertragung mit hoher Baudrate über die serielle Schnittstelle die Bearbeitung des ersten Interrupts nur die Zeit zur Übertragung eines Bytes in Anspruch nehmen.

Mit der Beendigung einer Interruptprozedur muß das entsprechende Bit im »ISR« gelöscht und damit die Interrupts niedrigerer Prioritäten wieder erlaubt werden. Dies geschieht mit der Übergabe von bestimmten Befehlen an den Interruptcontroller am Ende jeder Interruptprozedur. Es gibt drei Arten dieser sogenannten End-Of-Interrupt-Befehle (EOI-Befehle), die mit den verschiedenen Arbeitsmodi des Interruptcontrollers zusammenhängen (Tabelle 3). Im vollverschachtelten Modus beenden Sie die Interruptprozedur mit dem »nichtspezifischen EOI-Befehl«. Er bewirkt das Löschen des Bits im »ISR«, dessen Interrupt gerade bearbeitet wurde. Mit dem »spezifischen EOI-Befehl« löschen Sie ein ganz bestimmtes Bit im »ISR«, ohne auf das Prioritätenschema Rücksicht zu nehmen. Der Einsatz dieses EOI-Befehls ist nur mit genauer Kenntnis des 8259 und nur in bestimmten Fällen der Prioritätenverschiebung ratsam. Dazu später mehr.

Die dritte Alternative ist der »automatische EOI-Befehl«. Sobald ein Interrupt auftritt, setzt der Interruptcontroller selbständig das entsprechende ISR-Bit zurück. Der Einsatz dieses EOI-Befehls ist vor allem bei periodisch auftretenden Ereignissen angebracht, die sich nicht selbst unterbrechen. Jedem Interrupt ist im vollverschachtelten Modus des 8259 eine bestimmte Priorität zugeordnet. Das bedeutet aber, daß die Interrupts nicht gleichberechtigt behandelt werden. In einem anderen Modus, der »automatischen Rotation«, werden alle Interrupts gleichberechtigt behandelt. Nachdem ein Interrupt bearbeitet wurde, setzt der Interruptcontroller ihn auf die niedrigste Priorität. Sie können die Reihenfolge der Prioritäten also anders als üblich festlegen oder allen Interrupts die gleiche Priorität geben.

Bedeutung	Bit D5 bis D7 des OCW2
automatisches Rotieren löschen	000
nicht spezifischer EOI	001
keine Funktion	010
spezifischer EOI	011
automatisches Rotieren setzen	100
automatisches Rotieren nach nicht-spezifischem EOI	101
setzen der Priorität	110
spezifische Rotation	111

Tabelle 3. Die EOI- und Rotationsbefehle

Bit	Belegung
0	Timer
1	Tastatur
2	zweiter Controller
3	zweite serielle Schnittstelle
4	erste serielle Schnittstelle
5	zweiter paralleler Drucker
6	Diskettenlaufwerk
7	erster paralleler Drucker
0	Echtzeituhr
1 bis 4	frei
5	mathematischer Coprozessor
6	Festplatte
7	frei

Tabelle 2. Die Belegung der Signalleitungen bei den beiden Controllern beim AT

Priorität	7	6	5	4	3	2	1	0
ISR	0	0	0	0	0	0	0	0
Interrupt 6 angefordert								
ISR	0	1	0	0	0	0	0	0
Interrupt 3 angefordert								
ISR	0	1	0	0	1	0	0	0
Interrupt 3 fertig								
ISR	0	1	0	0	0	0	0	0
Interrupt 6 fertig								
ISR	0	0	0	0	0	0	0	0

Bild 4. Die Arbeitsweise der 8259 im vollverschachtelten Modus

Zur Unterstützung dieser automatischen Prioritätenrotation existieren zwei EOI-Befehle. Der »nichtspezifische EOI-Befehl« löst eine Rotation aus, beim »automatischen EOI-Befehl« rotiert der Interruptcontroller die Priorität. Die Prioritäten sind immer so verteilt, daß ein Interrupt die höchste Priorität besitzt und die übrigen Prioritäten nach links hin abnehmen. Was heißt das? Wie schon beschrieben, sind die Interrupts den einzelnen Bits der Register zugeordnet. Nach »links hin abnehmend« bedeutet beispielsweise, daß der dem Bit 4 zugeordnete Interrupt eine geringere Priorität hat als der dem Bit 5 zugeordnete.

Außer der permanenten Rotation der Prioritätsebenen können Sie das Prioritätenschema der Interrupts frei defi-

nieren. Beispielsweise können Sie dem Interrupt IRQ 0, der normalerweise die höchste Priorität besitzt, ohne weiteres die niedrigste Priorität zuordnen (Bild 5).

Um Kollisionen zu vermeiden, werden Prioritäten vergeben

In vielen Fällen ist es notwenig, ganz bestimmte Interrupts zu verbieten. Von der CPU her stehen Ihnen dazu die Befehle »cli« und »sti« zur Verfügung, die allerdings entweder alle Interrupts verbieten oder alle erlauben. Der 8259 kann auch einzelne Interrupts erlauben und verbieten. Dazu dient im vollverschachtelten Modus das Interrupt-Mask-Register (IMR). Mit dem Setzen oder Löschen des Bits, das der Interruptquelle zugeordnet ist, erlauben oder verbieten Sie den jeweiligen Interrupt. Jedoch sind im vollverschachtelten Modus alle niedrigen Interruptanforderungen unterdrückt, unabhängig vom Zustand des entsprechenden Bits im »IMR«. Erst nach dem Ende der aktuellen Interruptabarbeitung spielt das eine Rolle. War der Interrupt erlaubt, so wird er anschließend verarbeitet, andernfalls gänzlich ignoriert.

Bit	Bedeutung
D0 bis D2	Nummer der zu bearbeitenden Interruptebene
D3 bis D4	immer 0
D5 bis D7	EOI- und Rotationsbefehle (Tabelle 3)
D0 bis D1	Registerlesebefehl 10:»IRR« 11:»ISR«
D2	Pollbefehl
D3	immer 1
D4	immer 0
D5 bis D6	spezieller Maskenmodus 10: Maske löschen 11: Maske setzen
D7	immer 0

Tabelle 4. Der Aufbau der zweiten und dritten OCW

Im sogenannten »speziellen Maskenmodus« arbeitet der 8259 auf eine ganz andere Weise. Beim Bearbeiten einer Interruptanforderung unterdrückt der Controller automatisch alle Interrupts, auch die mit höherer Priorität. Nur ein Interrupt mit derselben Prioritätsstufe findet Beachtung.

Um einen Interrupt auszulösen, muß ein Gerät oder ein Baustein ein elektrisches Signal an den Interruptcontroller senden. Dieses elektrische Signal kann der Interruptcontroller in zwei unterschiedlichen Formen erkennen: als Flanken-Triggerung oder als Spannungs-Triggerung. Mit dem Erkennen eines elektrischen Signals werden die entsprechenden Informationen in den Registern des Interruptcontrollers aktualisiert, Sie können auf diese Informationen zugreifen und sie entsprechend auswerten.

Außer der Interruptverabeitung durch Unterbrechung eines Anwenderprogramms existiert noch das Polling von Interrupts. Nach dem Umschalten des

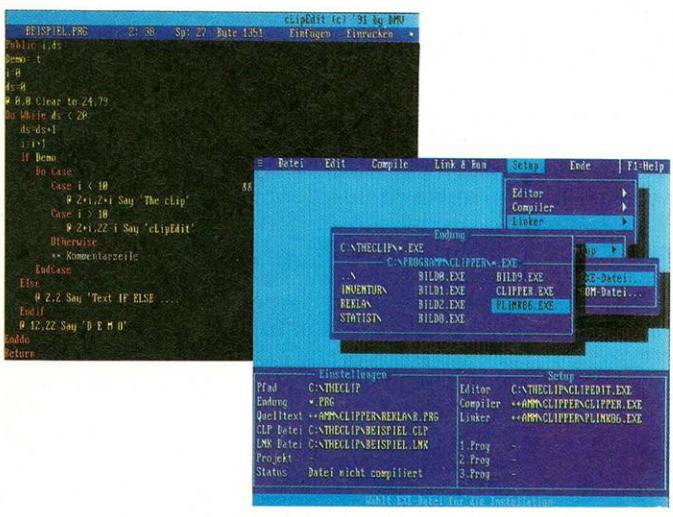
DMV-Integrator für Clipper

Moderne Programmiersprachen brauchen eine professionelle Entwicklungsumgebung

Der DMV-Integrator ist eine professionelle Entwicklungsumgebung, und zwar für Clipper®. Er vereinfacht und beschleunigt das Editieren, Compilieren und Linken Ihrer Programme, denn er faßt alle Programmaufgaben unter einer Oberfläche und für alle Clipper-Versionen zusammen.
Diese Leistungsmerkmale zeichnen den DMV-Integrator aus:

Die Programmierumgebung: Dateiauswahlbox • einstellbare Dateiendungen • compiliert und linkt Programme auf Knopfdruck • startet compiliertes Programm • Datei-Liste mit bis zu 10 Dateinamen für schnelles Laden • einfache Projektverwaltung • frei konfigurierbare Farbeinstellung • nutzt EMS-Speicher • arbeitet auch mit fremden Editoren zusammen • DOS-Shell • Online-Hilfe-System • Bedienung durch Pulldown-Menüs

Der Editor: • Wordstar®-kompatibler Editor • mit allen Standardfunktionen • automatisches Einrücken • Zeilenlänge bis 999 Zeichen • bis zu 6 Dateien werden gleichzeitig bearbeitet • Dateigröße nur durch freien Speicherplatz eingeschränkt • farbliche Hervorhebung von Clipper-Schlüsselworten • farbliche Kennzeichnung von Kommentarzeilen • informative Online-Hilfe



Systemkonfiguration: Festplatte, 640 KByte RAM, DOS ab 3.2 oder komp. System

DMV-Integrator für Clipper **199,— DM***

Informieren Sie sich auch über die

DMV-ProfiTools für Clipper **199,— DM***

*Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland DM 4,— bzw. für das Ausland DM 6,— Porto und Verpackung.

DMV-Verlag • Postfach 250 • 3440 Eschwege

DMV Daten- und Medienverlag

® Clipper ist eingetragenes Warenzeichen der Nantucket Corporation.
® Wordstar ist eingetragenes Warenzeichen MicroPro Corporation.

Interruptcontrollers in den Pollingmodus speichert er alle Informationen wie gewohnt. Allerdings unterbricht er kein Anwendungsprogramm mehr, sondern speichert die Informationen so lange, bis diese entweder gelesen oder durch eine erneute Interruptanforderung überschrieben werden.
Bei diesem Verfahren ist der Interruptcontroller mit einem Briefkasten vergleichbar, in den der Postbote einen oder mehrere Briefe (Interruptanforderungen) steckt. Wenn Sie (das Anwendungsprogramm) Zeit haben, schauen Sie nach, ob sich im Briefkasten Post befindet und entnehmen diese entsprechend ihrer Wichtigkeit (Priorität).
Im Gegensatz dazu sieht die weiter oben beschriebene Interruptverarbeitung durch Unterbrechung folgendermaßen aus: Der Postbote klingelt an der Haustür und gibt den Brief direkt ab. Sie (das Anwendungsprogramm) unterbrechen Ihre Arbeit, nehmen den Brief entgegen und lesen ihn. Falls während des Lesens des einen Briefes noch ein weiterer mit größerer Wichtigkeit (Priorität) ankommt, unterbrechen Sie das Lesen des ersten und lesen den neuen Brief.
Da der Interruptcontroller ein externes Bauteil ist, findet seine Programmierung über die für den PC üblichen I/O-Ports statt. Über diese I/O-Ports kommuniziert die CPU mit sämtlichen externen Geräten und Bauteilen. Hierzu stehen Ihnen als Entwickler vier Befehle zur Verfügung, die Daten in einen, durch »portnr« bestimmten, Port schreiben oder von ihm lesen:

- »outportb(portnr,byte)« schreibt ein Byte
- »inportb(portnr)« liest ein Byte
- »outportw(portnr,word)« schreibt ein Wort
- »inportw(portnr)« liest ein Wort

Bei der eigentlichen Programmierung des Interruptcontrollers 8259 sind zwei Phasen zu unterscheiden. Vor seiner Benutzung findet die Initialisierung durch die sogenannten »Initialisation Command Words« (ICWs) statt. Sie legen verschiedene Werte, wie zum Beispiel die Adressen der Interruptvektortabelle, fest. Während der Benutzung des Interruptcontrollers können Sie einige seiner Parameter durch die »Operational Command Words« (OCWs) modifizieren.
Jeder Interruptcontroller belegt jeweils zwei I/O-Ports, über die beide Phasen der Programmierung ablaufen. In den Beschreibungen zu den ICWs und OCWs finden Sie auch die dazugehörigen Portadressen für die Interruptcontroller.
Die erste Initialisierung des Interruptcontrollers findet schon beim Starten Ihres Rechners statt. Eine weitere Initialisierung wird für Sie erst dann notwendig, wenn Sie entscheidende Änderungen der Initialisierungsparameter vornehmen wollen. Je nachdem mit welchem Initialisierungswort Sie beginnen, legen Sie die Reihenfolge und Anzahl der darauffolgenden Initialisierungswörter fest.

IRQ	7	6	5	4	3	2	1	0
Priorität	6	5	4	3	2	1	0	7

Bild 5. Eine neue Prioritätsverteilung

- ICW 1 legt mit Bit 0 fest, ob ein ICW 4 benötigt wird oder nicht. Da der Interruptcontroller in Ihrem Rechner immer im 8086/8088-Modus arbeitet und dazu das ICW 4 benötigt, ist Bit 0 immer 1. Mit »D1« legen Sie fest, ob mehrere 8259 kaskadiert sind oder nicht. Falls dieses Bit nicht gesetzt ist, wird auch kein ICW 3 übergeben. Dieses Bit macht also nur bei ATs Sinn. »D2« ist im 8086/8080-Modus nicht von Bedeutung und wird auf 0 gesetzt. Mit »D3« legen Sie fest, ob der Interruptcontroller auf Signale mit Spannungs-Triggerung oder Flanken-Triggerung reagieren soll. Mit »D5« bis »D7« legen Sie die Adressen der Interruptvektoren im 80/85-Modus fest (beim PC bedeutungslos).
- ICW 2 bestimmt durch die oberen 5 Bits die Adresse der Interruptvektor-Tabelle. Im Normalfall ist hier der beim Starten übergebene Wert für den ersten Interruptcontroller gleich 8. Zu diesem Wert wird die jeweilige Interruptnummer addiert. »D1« und »D2« sind für den PC bedeutungslos.
- ICW 3 ist für die Kaskadierung mehrerer Interruptcontroller zuständig und wird nur übergeben, falls Bit »D1« des ICW 1 gesetzt war. Wenn ein Interruptcontroller als Master fungiert, sind die Bits dieses ICWs den Slave-Interruptcontrollern (nachgeordnete Controller) zugeordnet. Dabei steht jedes gesetzte Bit für einen als Slave angeschlossenen 8259. Ist der programmierte Interruptcontroller ein Slave, enthält dieses ICW in den ersten drei Bits seine Slave-Nummer.
- ICW 4 initialisiert den Arbeitsmodus des 8259. Bit 0 legt fest, mit welchem Prozessor (80/85, 8086 oder 8080) der Interruptcontroller zusammenarbeiten soll. Bit 1 ist für die Art der Beendigung der Interruptprozeduren verantwortlich. Beim PC wird normalerweise nach jeder Beendigung eines Interrupts ein nichtspezifischer EOI-Befehl von der Interruptprozedur an den 8259 gesendet. Bei gesetztem Bit 1 geschieht dies automatisch nach jedem Eintreten eines Interrupts. »D2« bis »D3« definieren die Pufferung des 8259 als Slave oder Master. »D4« legt den speziell verschachtelten Modus, der nur bei der Kaskadierung Verwendung findet, fest.

Während des Betriebs des Interruptcontrollers verändern Sie mit den folgenden drei OCWs die Parameter des Interrruptcontrollers (Tabelle 4).

- OCW 1 stellt die Maske der erlaubten und verbotenen Interrupts dar. Falls ein Bit gelöscht ist, ist dieser Interrupt erlaubt, während ein gesetztes Bit den entsprechenden Interrupt verbietet. Die jeweiligen Bitnummern dieses OCWs entsprechen dabei den Interruptnummern.
- OCW 2 ist für die Beendigung von Interruptprozeduren verantwortlich. Sie können die verschiedenen EOI-Befehle an den Interruptcontroller senden und die Art der Prioritätsrotation festlegen. Außerdem können Sie die Prioritäten grundsätzlich neu verteilen (siehe EOI-Befehle).
- OCW 3 ist für die Arbeit im Pollingbetrieb zuständig. Die Register »IRR« und »ISR« sind sowohl als Poll-Befehl als auch ohne besondere Wirkung auf den 8259 auslesbar. Des weiteren setzen oder löschen Sie mit diesem OCW die Maske für den speziellen Maskenmodus.

Während des Schreibens und Lesens auf die Ports eines Interruptcontrollers sollten alle Interrupts durch den Maschinensprachebefehl »cli« unterbunden werden. Andernfalls sind Nebeneffekte nicht auszuschließen. Sind die Operationen auf die Register durchgeführt, erlauben Sie die Interrupts durch den Maschinenbefehl »sti«.

Eine Funktionsbibliothek für den Interruptcontroller

Da bei der Programmierung von Interruptcontrollern eine ganze Reihe von Konventionen zu beachten ist, finden Sie hier eine kleine Funktionsbibliothek. Mit Hilfe von »hardint.c« (Listing 2) und der Kenntnis über die Grundlagen und die interne Struktur des 8259 entwickeln Sie ohne großen Aufwand Software, die den 8259 ausnutzt. Die Funktionen dieser Bibliothek arbeiten nahezu alle mit denselben Begriffen für die einzelnen Hardware-Interrupts (Tabelle 5). Dabei wird unterschieden zwischen Interrupts für PC und AT sowie nur für AT. Da beim AT zwei Interruptcontroller ihren Dienst verrichten, existieren auch mehr Hardware-Interrupts als beim PC mit nur einem 8259. Auch sind die internen Interruptnummern der Fesplatte bei PC und AT unterschiedlich. Dies alles löst die Funktionsbibliothek für Sie. Durch die Verwendung der in Tabelle 5 aufgeführten Begriffe greift die Bibliothek automatisch auf einen oder zwei Interruptcon-

troller zu und setzt oder löscht die für Ihren Rechner richtigen Interruptbits.

- »init_cont(adr1,adr2,hardint)« initialisiert den/die Interruptcontroller Ihres Rechners neu. Die beiden Adressen bestimmen die Speicherstellen, bei denen die jeweilige Interruptvektor-Tabelle eines Controllers beginnt: »adr1« für den ersten und »adr2« für den zweiten Interruptcontroller. Nach dem Starten ist die Adresse der Interruptvektoren für den ersten Interruptcontroller gleich 8 und für den zweiten (nur beim AT) gleich 70hex. Wollen Sie die Vektoren an eine andere Stelle verschieben, müssen Sie eine durch 8 teilbare Adresse angeben. Um diese Werte unverändert zu lassen oder deren ursprünglichen Zustand wiederherzustellen, setzen Sie in »adr1« und »adr2« den Wert »DEFAULT« ein. Für »hardint« setzen Sie die Hardware-Interrupts für die Initialisierung aus Tabelle 5 ein. Auch hier stellen Sie durch »DEFAULT« den Originalzustand wieder her. Mit

```
init_cont(0x20,DEFAULT,TIMER);
```

verschieben Sie die Interruptvektor-Tabelle des ersten Interruptcontrollers auf die Adresse 20hex und lassen nur den Timerinterrupt zu.

Mit folgendem Befehl stellen Sie den Originalzustand wieder her:

```
init_cont(DEFAULT,DEFAULT,DEFAULT);
```

Allgemein für PC und AT	nur AT
TIMER	CASCADE
KEYBOARD	LPT2
COM2	REALTIMEC-LOCK
COM1	MATHCOP
FLOPPY	
LPT1	
Sind alle Interrupts für PC und AT gemeint, dann ist der Parameter »ALL« zu verwenden	

Tabelle 5. Die Hardware-Interrupts der Bibliotheksfunktionen

- »set_prio(hardint)« verschiebt das Prioritätenschema so, daß der angegebene Interrupt die höchste Priorität einnimmt. Die anderen Prioritäten fallen in Richtung der absteigenden Bitnummer der Maske ab. Im Normalfall besitzt der Timerinterrupt die höchste Priorität. Mit dem Befehl

```
set_prio(KEYBOARD);
```

weisen Sie dem Tastaturinterrupt die höchste Priorität zu.

- »forbid_int(hardint)« verbietet einzelne oder alle Hardware-Interrupts. Mit der Anweisung

```
forbid_int(TIMER¦KEYBOARD);
```

verbieten Sie das Auftreten von Timer- und Tastaturinterrupts.

- »permit_int(hardint)« ist das Gegenstück zu »forbid_int« und erlaubt

Bit	Bedeutung
S	Startbit
D0 bis D7	zu übertragendes Datum mit einer Länge von 5 bis 8 Bit
P	Paritätsbit (optional)
St	Stopbit
St	Stopbit (optional)

Tabelle 6. Das Format eines seriell übermittelten Bytes

Sesam öffne dich!

BCi DisAsm erschließt jedes Programm!

Wandelt den Programmcode in ein dokumentiertes Assemblerlisting um. Mit normalen Editoren können Sie nun verändern und modifizieren, und mit gängigen Assemblern und Debuggern kann danach wieder assembliert werden.

Es gibt kaum einen, der BCi DisAsm das Wasser reichen kann!

BCi DisAsm i86
Intelligenter Disassembler Version 1.0

Auch Experten kennen nicht immer die Lösung, aber sie wissen, wo sie suchen müssen.
BCi DisAsm hilft JEDEM, die verwirrenden und frustrierenden Software-Puzzles, welche sich beim Umgang mit dem Computer ergeben, zu lösen. Und zwar schneller als mit jedem anderen Produkt, das Sie kaufen können, egal, um welche Preisklasse es sich handelt.
Mit dem BCi DisAsm erwerben Sie sofort ein Know-how, daß Sie normalerweise Jahre an Erfahrung kosten würde. Versuchen Sie nicht, das Rad neu zu erfinden!

BCi DisAsm – Der intelligente Disassembler
BCi DisAsm ist ein intelligenter, selbsttätig dokumentierender MS-DOS-Disassembler. Die ausgefeilten Routinen im BCi DisAsm analysieren in kürzester Zeit selbst komplexen und umfangreichen Programmcode und zeigen dabei potentielle Problempunkte auf.

BCi DisAsm – Das Lernmittel
Sie brauchen kein Spezialist in Assembler-Programmierung zu sein. BCi DisAsm produziert aus jedem beliebigen Programm ihres IBM-kompatiblen Computers ein gut dokumentiertes, leicht verständliches Assemblerlisting.

BCi DisAsm – Einfach in der Handhabung
BCi DisAsm kann sowohl über Parameter von der DOS-Kommandozeile aus als auch interaktiv über Menüs gesteuert werden. Die Menüstruktur ist Lotus-kompatibel und selbstverständlich in Deutsch. Es besteht keine Notwendigkeit, neue Befehle zu lernen oder ständig im Handbuch nachzuschlagen. BCi DisAsm ist unmittelbar anwendbar!

Minimale Systemanforderungen
256 kByte RAM, 8088/8086/80186/80286 oder 80386 CPU, MS-DOS oder PC-DOS Version 2.0 oder größer

BCi DisAsm wird Ihnen bei den folgenden Schritten helfen:

1. Lernen Sie Assembler-Programmierung, wenn Sie möchten.
2. Entdecken Sie, warum Programm XYZ auf Ihrem Rechner nicht läuft, und tun Sie etwas dagegen.
3. Ändern oder entfernen Sie Programmteile.
4. Sichern Sie sich gegen destruktive Programme ab. Finden Sie einen "Virus", und entfernen Sie ihn.
5. Integrieren Sie Code aus compilierten Programmen in eigene Anwendungen.
6. Erhöhen Sie die Kompatibilität Ihrer Software.
7. Passen Sie Software an verschiedene MS-DOS-Versionen an.

© 1987 BCi, West Germany und USA
Lotus ist eingetragenes Warenzeichen der Lotus Corp.
MS-DOS ist eingetragenes Warenzeichen der Microsoft Corp.

BCi DisAsm kostet inkl. eines sehr ausführlichen deutschen Handbuchs (mehr als 100 Seiten) nur

199,– DM (unverbindliche Preisempfehlung)
Wenn Sie über den DMV-Bestellservice bestellen, gilt folgendes:

Inland:		Ausland:	
Einzelpreis	199,– DM	Einzelpreis	199,– DM
zzgl. Versandkosten	4,– DM	zzgl. Versandkosten	6,– DM
Endpreis	**203,– DM**	**Endpreis**	**205,– DM**

Bitte benutzen Sie die Bestellkarte.

DMV-Verlag · Postfach 250 · 3440 Eschwege

DMV Daten- und Medienverlag

einzelne oder alle Hardware-Interrupts.

```
permit_int(COM1¦COM2¦FLOPPY);
```

erlaubt Interrupts durch die beiden seriellen Schnittstellen und die Diskettenstation.

- »read_reg(REG)« liefert als Rückgabewert den Inhalt eines der drei Interruptcontroller-Register, das Sie mit dem Parameter »REG« spezifizieren. Der zurückgelieferte Wert ist ein Wort, dessen niederwertiges Byte den ersten Interruptcontroller und dessen höherwertiges Byte den zweiten Controller betrifft. Mit dem Befehl

```
printf("%x\n", reg_reg(IRR));
```

erhalten Sie beim AT den Inhalt der »IRRs« des ersten und zweiten Interruptcontrollers.

- »EOI_not_spec« sendet einen nichtspezifischen EOI-Befehl an den Interruptcontroller und zeigt ihm dadurch an, daß der entsprechende Interrupt bearbeitet wurde und das entsprechende Bit im »ISR« wieder freigegeben werden kann.

COM1	COM2	Register	DLAB
03F8hex	02F8hex	TX-Puffer	0
03F8hex	02F8hex	RX-Puffer	0
03F8hex	02F8hex	Divisor-Latch, Lowbyte	1
03F9hex	02F9hex	Divisor-Latch, Highbyte	1
03F9hex	02F9hex	Interrupt-Enable-Register	0
03FAhex	02FAhex	Interrupt-Identifikation-Register	–
03FBhex	02FBhex	Line-Control-Register	–
03FChex	02FChex	Modem-Control-Register	–
03FDhex	02FDhex	Line-Status-Register	–
03FEhex	02FEhex	Modem-Status-Register	–

Tabelle 8. Die Register und Portadressen des 8250

Daten über die serielle Schnittstelle bitweise übertragen

Nahezu jeder Rechner ist mit mindestens einer Schnittstelle ausgestattet, die zur seriellen Übertragung von Daten dient. Beim PC verwaltet der Baustein 8250 diese Schnittstelle. Vor der Beschreibung dieses Prozessors und seiner Funktionsweise erfahren Sie zunächst, auf welche Weise eine serielle Datenübertragung abläuft.

Um Daten von einem Rechnersystem zu einem anderen zu übertragen, ist eine physikalische Verbindung beider Rechner notwendig. Im Falle der seriellen Schnittstelle benötigen Sie ein sogenanntes X-Modem-Kabel. Bei der Datenübertragung sendet der eine Rechner das Datum über die Verbindungsleitung zum anderen Rechner. Daß dies nicht ganz so einfach ist, wird allein dadurch klar, daß die CPU keinen direkten Zugriff auf diese Verbindungsleitung hat (von Transputern und ihren Links einmal abgesehen).

Zwischen der CPU und der Verbindungsleitung fungiert deshalb ein serieller Übertragungbaustein als Mittler. Um ein Datum zu übertragen, schickt die erste CPU dieses Datum an den seriellen Baustein ihres Rechners. Dieser Baustein nimmt über die Verbindungsleitung Kontakt mit dem seriellen Baustein des anderen Rechners auf und übergibt diesem das Datum. Der Empfängerbaustein im zweiten Rechner gibt das übertragene Datum an seine CPU weiter. Die Geschwindigkeit der Übertragung hängt von der Anzahl der Übertragungsleitungen ab. Um ein Byte auf einmal zu übertragen, benötigen Sie mindestens acht Sendeleitungen: für jedes Bit eine Leitung. Das bedeutet aber, daß auch achtmal so hohe Kosten für die Übertragungsleitungen und Mittlerbausteine anfallen – gegenüber nur einer einzigen Sendeleitung. Aus Kostengründen besitzt jeder Kommunikationspartner bei der seriellen Übertragung nur eine Sende- und eine Empfangsleitung. Der sendende serielle Baustein teilt ein zu übertragendes Byte in seine Bits auf und überträgt diese einzeln über die eine Leitung an den anderen seriellen Baustein, der die Bits wieder zu einem ganzen Byte zusammensetzt. Danach kann die CPU das Byte aus dem seriellen Baustein lesen.

Grundsätzlich müssen bei der Übertragung von Daten beide Übertragungspartner wissen, wann eine Informationseinheit übertragen wird und wie lange diese auf den Leitungen ablesbar sind. Hierbei wird zwischen der synchronen und asynchronen Datenübertragung unterschieden.

Bei der synchronen Datenübertragung existiert eine Taktleitung, die Übertragungsbeginn und -dauer für beide Übertragungspartner anzeigt. Man sagt, beide Übertragungspartner sind sychronisiert oder laufen synchron zueinander. Bei der seriellen Schnittstelle handelt es sich jedoch um eine asynchrone Übertragungseinrichtung. Es existiert also keine gesonderte Taktleitung, sondern jedem Byte geht ein sogenanntes Startbit voran, das die Übertragung einleitet. Mit diesem Startbit wird ein interner Taktgenerator des seriellen Bausteins gestartet und das ankommende Byte (das heißt, die einzelnen Bits) in diesem Takt empfangen. Dabei ist wichtig, daß bei Sender und Empfänger dieselbe Taktrate eingestellt ist. Weiterhin werden pro übertragenem Byte optional ein Paritätsbit sowie ein oder zwei Stopbits übertragen (Tabelle 6). Diese zusätzlichen Informationen erleichtern dem seriellen Baustein das Erkennen von Übertragungsfehlern.

Pinnummer	Name	Bedeutung
2	TD	(Transmit Data) Sendeleitung: Über diese Leitung werden die Datenbits übertragen
3	RD	(Receive Data) Empfangsleitung: Über diese Leitung werden die Datenbits empfangen
4	RTS	(Request To Send) Sendeanfrage: Der Sender signalisiert dem Empfänger den Wunsch zur Datenübertragung
5	CTS	(Clear To Send) Sendebereitschaft: Der Empfänger signalisiert die Bereitschaft zum Empfang von Daten
6	DSR	(Data Set Ready) Empfänger angeschlossen: Das DCE signalisiert dem DTE, daß sich ein Übertragungspartner am Verbindungende befindet
20	DTR	(Data Terminal Ready) Sender angeschlossen: Das DTE signalisiert, daß es an die Übertragungsleitung angeschlossen ist
22	RI	(Ring Indicator) Telefon klingelt: Falls das DCE ein Modem ist, Anzeige des Klingelzeichens
8	CD	(Carrier Detect) Trägerfrequenz endeckt: Bei der Datenübertragung mit Modems ist während der gesamten Kommunikation zwischen ihnen eine solche Trägerfrequenz notwendig

Tabelle 7. Signalpins der seriellen Schnittstelle

Außer den Sende- und Empfangsleitungen sind noch einige Steuerleitungen überaus wichtig. Die serielle Schnittstelle Ihres Rechners enthält die Steuerleitungen, die nach dem RS232-Standard notwendig sind. Der RS232-Standard beschreibt die wichtigsten Merkmale zum Datenaustausch zwischen sogenannten DTEs (Data Terminal Equipment) und DCEs (Data Control Equipment). Das DTE stellt einen Rechner dar, der mit dem DCE (einem Modem) kommuniziert. Die eigentliche Kommunikation über Langstrecken läuft nur zwischen zwei DCEs ab. Der RS232-Standard legt fest, unter welchen Bedingungen und mit welcher Signalreihenfolge Daten übertragen werden.

Zunächst lernen Sie die Signale kennen, die der RS232-Standard benötigt. An Ihrem Rechner finden Sie einen 25poligen Anschluß (manchmal aber auch nur 9polig), der direkt mit einem seriellen Baustein verbunden ist. Da nicht alle Pins dieser Schnittstelle tatsächlich belegt sind, zeigt Tabelle 7 die Bedeutung der belegten Pins. Pin 1 liegt oben rechts und Pin 25 unten links.

Bei der Datenübertragung unterscheidet man zwischen dem Vollduplexbetrieb, bei dem beide Kommunikationspartner in der Lage sind, Daten zu senden und zu empfangen, und dem Halbduplexbetrieb, bei dem immer nur einer Sender und einer Empfänger ist. Dieser Halbduplexbetrieb ist allerdings ein Relikt aus den Anfängen der Datenübertragung und findet heute keine Verwendung mehr (außer bei der Post).

Bit	Bedeutung	Inhalt	Wirkung
0 bis 1	Zeichenlänge	00	5 Bit Länge
		01	6 Bit Länge
		10	7 Bit Länge
		11	8 Bit Länge
2	Anzahl der Stopbits	0	1 Stopbit
		1	2 Stopbits
3	Parität	0	keine Parität
		1	Paritätsprüfung und -erzeugung
4	gerade/ungerade	0	ungerade Parität
5	feste Parität	0	keine feste Parität
		1	wenn Bit 4 = 0, dann Paritätsbit immer 1
			wenn Bit 4 = 1, dann Paritätsbit immer 0
6	Break	0	Break aus
		1	Break an
7	DLAB	0	Zugriff auf TX-Puffer, RX-Puffer und »IER«
		1	Zugriff auf Divisor-Latch-Register

Tabelle 9. Das Line-Control-Register (LCR)

Ein Protokoll regelt beim RS232-Standard die Übertragung

Zum RS232-Standard gehört außerdem ein sogenanntes Protokoll. Ein Protokoll legt fest, welche Reihenfolge der Signale die Übertragungspartner beim Senden und Empfangen von Daten einhalten müssen. Stellen Sie sich vor, ein Sender schickt ohne Vorwarnung eine Reihe von Daten, und der Empfänger befindet sich zu diesem Zeitpunkt in der Bearbeitung einer kritischen Prozedur, in der er keine Daten empfangen kann. In diesem Fall sind die Daten nicht einfach nur verloren, sondern:

- der Sender weiß nicht, daß der Empfänger die Daten nicht empfangen hat, und
- der Empfänger weiß nicht, daß Daten gesendet wurden.

Die Übertragung mit einem Protokoll vermeidet diese Inkonsistenzen gänzlich. Im RS232-Protokoll geschieht dies auf folgende Weise: Beide Übertragungspartner zeigen durch das jeweilige Setzen des DTR-Signals an, daß Sie miteinander verbunden sind. Will einer der beiden Senden, setzt er das Signal »Ready To Send« (RTS) und wartet auf die Antwort des Empfängers. Der Empfänger hingegen kann nun alle Arbeiten beenden und mit dem Senden des Signals »Clear To Send« (CTS) dem Sender seine Empfangsbereitschaft anzeigen. Nachdem der Sender das CTS-Signal empfangen hat, beginnt er mit der Übertragung der eigentlichen Daten. Danach schaltet der Sender sein RTS-Signal ab, worauf der Empfänger mit dem Abschalten des CTS-Signals reagiert – die Übertragung ist komplett.

Die Pins der seriellen Schnittstelle sind in Ihrem Rechner mit dem bereits erwähnten Asynchronous Communicatios Element (ACE) 8250 verbunden. Dieser Baustein übernimmt sowohl die Serialisierung der Daten als auch das Hinzufügen von Start-, Stop- und Paritätsbits. Im Normalfall besitzt ein Rechner wenigstens ein, meistens aber zwei ACEs. Mit einer speziellen Multiportkarte können Sie weitaus mehr serielle Schnittstellen an einen einzigen Rechner anschließen. Dies wird besonders beim Multiuserbetrieb interessant, wo jede serielle Schnittstelle gleichzeitig einen neuen Arbeitsplatz bedeutet. Die Programmierung des ACE findet genauso wie beim Interruptcontroller 8259 über die I/O-Ports des Rechners und mit den bereits beschriebenen Befehlen »inportb«, »outportb«, »inportw« und »outportw« statt. Jeder 8250 belegt insgesamt sieben I/O-Ports. In Tabelle 8 sind die zehn Register eines 8250 und ihre Verteilung auf die sieben I/O-Ports aufgelistet. Durch das sogenannte DLAB-Bit schalten Sie einen einzigen Port auf verschiedene Register und sprechen so mehr Register an als Ports zur Verfügung stehen.

Ähnlich wie beim Interruptcontroller sind auch beim 8250 zwei Phasen der Programmierung zu unterscheiden: die Initialisierung des Bausteins und der Betrieb. Bei der Initialisierung werden die Basisparameter festgelegt, die für eine spätere Kommunikation von besonderer Wichtigkeit sind. Dies betrifft vor allem die Länge der Zeichen, die Anzahl der Stoppbits, die Übertragungsrate (Baudrate) und die Parität der Übertragung. Diese Parameter sind über das sogenannte Line-Control-Register (LCR) zu setzen (Tabelle 9).

Bei der Parität gibt es insgesamt vier Alternativen. Die Steuerlogik erzeugt:

- eine gerade Anzahl gesetzter Bits,
- eine ungerade Anzahl gesetzter Bits,
- ein auf 1 gesetztes Paritätsbit und
- ein auf 0 zurückgesetztes Paritätsbit.

Die Baudrate legen Sie beim ersten seriellen Baustein ihres Systems auf folgende Weise fest: Zunächst setzen Sie das DLAB-Bit des »LCR« mit dem Befehl

```
outportb(0x3fb,
        inportb(0x3fb)¦0x80);
```

auf 1. Danach berechnen Sie den Wartewert, den der Baudraten-Generator zu warten hat, bevor ein neues Bit übertragen wird, nach der Formel

```
Wartewert = 115200 / Baudrate
```

Dieser Wartewert wird in ein unteres und ein oberes Byte aufgeteilt und in die Divisor-Latch-Register eingetragen. Damit steht die neue Übertragungsrate des seriellen Bausteins fest. Interessant ist dabei, daß durch diese Formel

AB SOFORT IM HANDEL!

Die neue WinDOS bietet jetzt umfassende und qualifizierte Information rund um das brandaktuelle Thema Windows:

- News
- Software
- Werkstätten
- Tips & Tricks
- Marktübersichten
- Programmierung
- und vieles, vieles mehr.

Windows besser verstehen, beherrschen und programmieren. Mit WinDOS.

In diesem Heft:
Diskette mit allen Beispielprogrammen plus WinBasic-Demo

DMV-Verlag • Postfach 250 • 3440 Eschwege

DMV Daten- und Medienverlag

Bit	Bedeutung
0	Daten angekommen
1	Senderegister leer
2	Wechsel des Leitungszustands
3	Wechsel des Modemzustands
4 bis 7	immer Null

Tabelle 10. Das Interrupt-Enable-Register (IER)

Bit	Bedeutung
0	DTR: Wenn gesetzt, wird DTR-Signal erzeugt
1	RTS: Wenn gesetzt, wird RTS-Signal erzeugt
2	keine Funktion
3	OUT2: Das Setzen schaltet die Leitung zum Interruptcontroller frei
4	LOOP: Selbsttest des 8250, wobei dieser seine Ausgänge mit den Eingängen verbindet
5 bis 7	nicht verwendet

Tabelle 11. Das Modem-Control-Register (MCR)

Bit	Bedeutung
0	Interrupt durch 8250
1 bis 2	00 Wechsel im Modemzustand 01 Senderegister leer 10 Empfängerdaten angekommen 11 Wechsel des Leitungszustandes
3 bis 7	keine Funktion

Tabelle 12. Das Interrupt-Identification-Register (IIR)

Bit	Bedeutung	Löschen des Bits
0	Daten angekommen	Lesen des RX-Puffers
1	Überlauffehler	Lesen des »LSR«
2	Paritätsfehler	Lesen des »LSR«
3	Framefehler	Lesen des »LSR«
4	Breakinterrupt	Lesen des »LSR«
5	TX-Puffer leer	Schreiben eines Zeichen in den TX-Puffer oder Lesen des »IIR«
6	interner Sendepuffer leer	wird von TX-Puffer geladen
7	keine Funktion	

Tabelle 13. Das Line-Status-Register (LSR)

Bit	Bedeutung	Funktion
0	DCTS	Wechsel des CTS-Signals
1	DDSR	Wechsel des DSR-Signals
2	TERI	Wechsel des RI-Signals
3	DCD	Wechsel des CD-Signals
4	CTS	Zustand der CTS-Leitung
5	DSR	Zustand der DSR-Leitung
6	RI	Zustand der RI-Leitung
7	CD	Zustand der CD-Leitung

Tabelle 14. Das Modem-Status-Register (MSR)

auch das Maximum der Baudrate festgelegt ist. Sie beträgt 115200 Baud. In diesem Fall wartet der Baustein nämlich genau (115200/115200 = 1) einen Takt lang, um ein Bit vom anderen zu trennen. Kürzer geht es nicht.
Zur Koppelung des 8250 mit dem Interruptcontroller 8259 müssen Sie zusätzlich das dafür verantwortliche Register »IER« mit dem Wert initialisieren, der die Interrupts zuläßt, die Sie wünschen (Tabelle 10). Dabei kann der 8250 einen Interrupt durch folgende vier Ereignisse veranlassen:

- Daten wurden empfangen.
- eine Änderung des Leitungszustandes wurde festgestellt.
- das Senderegister ist leer.
- eine Änderung des Modemzustandes ist eingetreten.

Die Leitung vom seriellen Baustein zum Interruptcontroller 8259 müssen Sie freischalten. Dies geschieht durch das Setzen des OUT2-Bits im Modem-Control-Register (MCR) (Tabelle 11). Nun ist der serielle Baustein so an den Interruptcontroller gekoppelt, daß an der seriellen Schnittstelle auftretende Ereignisse einen Sprung der CPU in eine Interruptprozedur auslösen. Jetzt müssen Sie nur noch die entsprechenden Interruptvektoren auf eine für die Kommunikation verantwortliche Prozedur setzen. Für die erste serielle Schnittstelle ist dies der Vektor 0Chex, während die zweite Schnittstelle den Vektor 0Bhex belegt.

Das Line-Status-Register zeigt Ihnen Übertragungsfehler an

Mit dem Ankommen von Daten an der seriellen Schnittstelle wird nach der Initialisierung des ensprechenden Vektors Ihre Interruptroutine automatisch ausgeführt. Die Aufgabe Ihrer Funktion ist es nun, die ankommenden Zeichen anzunehmen und zu verarbeiten. Dazu ist der Zugriff auf verschiedene Informationen notwendig. Zunächst sind Sie in der Lage, durch das Lesen des Interrupt-Identification-Registers (IIR) festzustellen, welches Ereignis den Interrupt ausgelöst hat (Tabelle 12).
Häufig kommt es vor, daß ein Übertragungsfehler aufgetreten ist. Ob dies der Fall ist und welcher Fehler aufgetreten ist, stellen Sie durch das Lesen des Line-Status-Registers (LSR) fest (Tabelle 13). Dieses Register enthält immer den aktuellen Zustand der Übertragungsleitung. Bei vielen Protokollen, beispielsweise dem RS232-Protokoll, ist es notwendig, die Signale »DTR«, »DSR«, »RTS« und »CTS« auszutauschen. Für das Setzen der eigenen Signale, ist das schon erwähnte Register »MCR« zuständig. Den Zustand der Signalleitungen des Übertragungspartners ermitteln Sie durch das Lesen des Modem-Status-Registers (MSR) (Tabelle 14).
Wenn keine Störung vorliegt, finden Sie im RX-Puffer ein neu angekommenes Zeichen, das Sie durch Lesen aus diesem Register entfernen. Im Normalfall sollten Sie die ankommenden Zeichen in einem ausreichend großen Puffer zwischenspeichern, da die Weiterverarbeitung meist mehr Zeit in Anspruch nimmt, als zwischen der Annahme zweier Zeichen verstreicht.
Eine weitere Methode, Zeichen über die serielle Schnittstelle zu empfangen, ist der Pollingbetrieb. Dieser läuft wie beim Interruptcontroller ab. Der Pollingbetrieb ist allerdings für viele Anwendungen, beipielsweise Terminalprogramme und Netzwerke, völlig ungeeignet. Sobald ein Anwendungsprogramm nicht rechtzeitig nach Empfang eines Zeichens den Empfangspuffer ausliest, überschreibt das nächste ankommende Zeichen diesen, und die Daten gehen verloren. Da aber ein Anwendungsprogramm auf längere Zeit unterbrochen werden kann (TSR-Programm, Interruptaufruf), ist dieser Fehler geradezu vorprogrammiert. Deshalb ist diese Art der seriellen Übertragung nicht ratsam und nur in bestimmten Fällen anwendbar.
Durch das Schreiben eines Zeichens in den TX-Puffer wird es automatisch über die serielle Schnittstelle gesendet. Der 8250 nimmt dabei keine Rücksicht darauf, ob er ein DSR-Signal empfängt oder nicht. Das Signal-Handshaking und Protokoll ist allein Sache des Entwicklers. Der legt fest, welche Signale vorhanden sind oder vor dem Senden eines Zeichens ausgetauscht werden müssen. Ein Beispiel für solch ein Protokoll haben Sie schon mit dem RS232-Protokoll kennengelernt.
Um Ihnen die Programmierung der seriellen Schnittstelle zu erleichtern, finden Sie auch dafür eine Funktionsbibliothek.
Da Sie mit Ihrem Rechner mehr als zwei serielle Schnittstellen verwalten können, liegt den Funktionen der hier vorgestellten Bibliothek »serial.c« (Listing 3) eine Struktur für jede zu verwaltende Schnittstelle zugrunde. Mit dieser Struktur (Tabelle 15) greifen Sie sehr flexibel auf beliebig viele serielle Schnittstellen zu. Sie müssen diese vor der eigentlichen Kommunikation lediglich initialisieren.
Für die zwei üblichen seriellen Schnittstellen füllen Sie diese Struktur sehr einfach über vordefinierte Werte aus. Für die erste Schnittstelle gilt:

```
com.p_adr = COM1_BASE;
com.int_nr = 0x0c;
com.hard_int= COM1;
```

und für die zweite Schnittstelle:

```
com.p_adr = COM2_BASE;
com.int_nr = 0x0b;
com.hard_int= COM2;
```

Bei weiteren seriellen Schnittstellen müssen Sie selbständig die belegten Interruptvektoren und die entsprechenden Interruptcontroller-Bits und Portadressen eintragen. Bei allen folgenden Funktionen stellt der Parameter »COM« stets einen Zeiger auf eine solche initialisierte COM-Struktur dar.

- »init_com(COM,baud,datelen,stopbits,parity,ier,mcr)« initialisiert die durch die COM-Struktur definierte Schnittstelle. Mit dem Longwert »baud« legen Sie die Baudrate zwischen 0 und 115200 Baud fest. Die Länge der übertragenen Zeichen darf zwischen 5 und 8 liegen; Sie legen sie über den Parameter »datelen« fest. Der übergebene Wert von »stopbits« ist entweder 0 oder 1. Bei »parity« können Sie zwischen »NONE«, »ODD« oder »EVEN« wählen. Mit dem Parameter »ier« bestimmen Sie die Ereignisse, bei denen ein Interrupt ausgelöst werden soll (»DATA_REC«, »THR_EMPTY«, »MODEM_CHG«, »LINE_CHG«). In »mcr« übergeben Sie den Zustand des Modem-Control-Registers (»DTR«, »OUT2«, »RTS«). Diese Prozedur sichert außerdem die alten Werte des Interruptvektors, des »IER« und des »MCR«.
- Mit »set_para(COM.p_adr,baud, datelen,stopbits,parity)« ändern Sie auch während des Betriebs die Einstellungen der Übertragungsparameter. Die Parameter entsprechen denen der Prozedur »init_com«.
- »exit_com(COM)« beendet die Arbeit mit der angesprochenen seriellen Schnittstelle und stellt den früheren Zustand des Interruptvektors und der Register »IER« und »MCR« wieder her.

Das Beispiel »communic.c« (Listing 4) installiert mit den gerade erklärten Funktionen eine neue Interruptprozedur »new_ser« für die zweite serielle Schnittstelle. Diese Prozedur »new_ser« empfängt ankommende Daten und speichert sie sofort in einem Ringpuffer, der mit den Funktionen »push_ch« und »pop_ch« aus »queue.c« (Listing 5) verwaltet wird.

Die sieben Schichten zur erfolgreichen Datenübertragung

Durch die Funktion »sendchar(ch, COM)« sendet ein Programm das Byte »ch« an den anderen Rechner.
Gerade bei der asynchronen Übertragung von Daten mit hohen Baudraten sind Übertragungsfehler an der Tagesordnung. Daher ist es wichtig, durch verschiedene Prüfstationen die Richtigkeit der übertragenen Daten abzusichern. Zu diesem Zweck wurde das »Open System Interconnection Modell« (OSI) entwickelt. Diese Norm umfaßt ingesamt sieben aufeinander

Typ	Name	Bedeutung
word	p_adr	Basisadresse der I/O-Ports
word	int_nr	Interruptvektor-Nummer
word	hard_int	Interruptcontroller-Bit
void interrupt*	int_vect	Adresse der neuen Interruptprozedur

Tabelle 15. Die Struktur zur Verwaltung der seriellen Schnittstelle

aufbauende Schichten zur einwandfreien Übertragung von Daten. Falls Sie also eine professionelle Anwendung entwickeln, ist die Einhaltung dieses Standards ein großer Schritt für das erfolgreiche Funktionieren Ihres Sofwareprodukts.

- Die erste Schicht heißt »Physical Layer« und beschreibt die eigentliche physikalische Verbindung zweier Rechner. Alle darüberliegenden Schichten sind rein logischer Natur. Allein diese Schicht ist letztendlich für den Austausch der Daten verantwortlich.
- Für das An- und Abmelden der Übertragung durch Signale ist »Link Layer«, die sogenannte Übermittlungsschicht, zuständig. Im hier vorgestellten Beispiellisting wird gerade diese Schicht erreicht. Durch die Übertragung von sogenannten Frames werden schon auf dieser Ebene Fehler erkannt und behoben.
- Mit »Network Layer« ist die dritte Schicht erreicht. Hier werden logische Kanäle auf die Übertragung aufgesetzt. Damit kann eine physikalische Verbindung viele Kanäle mehrerer Anwendungen unabhängig verwalten.
- Die vierte Schicht nennt sich »Transport Layer« und ist für einen zuverlässigen und kostengünstigen Datentransport zuständig. Dazu gehört auch das Finden des schnellsten Weges durch ein Netzwerk mit vielen Knoten. In dieser Schicht siedeln Sie eventuelle Datenkompressions-Algorithmen an.
- Für die Kommunikationstransparenz von Prozessen auf verschiedenen Rechnersystemen ist »Session Layer« verantwortlich.
- In der Transformationsschicht »Presentation Layer« werden Transformationen der übertragenen Daten vorgenommen. Beispielsweise könnte der abschließende ASCII-Wert 13 durch eine Folge der ASCII-Werte 13 und 10 ersetzt werden.
- Die letzte und oberste Schicht ist »Application Layer«. Mit ihr arbeitet der Entwickler der Software bei Datenübertragung und stellt Sie dem Anwender zur Verfügung.

Die Aufgaben der sieben Schichten wurden hier nur sehr oberflächlich angesprochen, da eine ausführliche Erklärung den zur Verfügung stehenden Rahmen sprengen würde. Dieser Abschnitt soll Ihnen vor allem vor Augen führen, wieviel Wert Sie bei einer Anwendung mit Datenübertragung auf deren Korrektheit legen müssen.

(Matthias Oberdorfer/rm)

```
[A53]    1: /*Datei: compatib.c
[HW6]    2:   Funktion: Kompatibilitätsbox zwischen
[ZØ4]    3:             Turbo C und Microsoft C
[MP1]    4:   Sprache: C
[3L4]    5:   Autor: Matthias Oberdorfer
[T15]    6:   (c)1991 DMV Widuch GmbH & Co. KG */
[ØØØ]    7:
[HR2]    8: #ifndef COMPATIB
[RR2]    9: #define COMPATIB
[7M2]   1Ø: #include<dos.h>
[ØØØ]   11:
[L72]   12: #ifndef MK_FP
[F25]   13: #define MK_FP(s,o) ((void far *)\
[7R4]   14:           (((unsigned long)(s)<<16)|(o)))
[NZØ]   15: #endif
[ØØØ]   16:
[W64]   17: /*  Microsoft compatibel  */
[ØØØ]   18:
[KX2]   19: #ifdef __TURBOC__
[UZ4]   2Ø: #define PUSHF    __emit__(Øx9c)
[GW4]   21: #define POPF     __emit__(Øx9d)
[KY4]   22: #define _chain_intr(adr){ \
[3K3]   23:   __emit__(Øx5Ø,Øx5Ø); \
[5V3]   24:   STCKPTR(Ø)=STCKPTR(4); \
[BV3]   25:   STCKPTR(2)=STCKPTR(6); \
[HV3]   26:   STCKPTR(4)=STCKPTR(8); \
[E14]   27:   STCKPTR(6)=STCKPTR(1Ø); \
[P14]   28:   STCKPTR(8)=STCKPTR(12); \
[E84]   29:   STCKPTR(1Ø)=STCKPTR(14); \
[U84]   3Ø:   STCKPTR(12)=STCKPTR(16); \
[694]   31:   STCKPTR(14)=STCKPTR(18); \
[P84]   32:   STCKPTR(16)=STCKPTR(2Ø); \
[794]   33:   STCKPTR(18)=FP_OFF(adr); \
[P84]   34:   STCKPTR(2Ø)=FP_SEG(adr); \
[NT5]   35:   __emit__(Øx5d,Øx5f,Øx5e,Øx1f,ØxØ7,\
[Ø94]   36:          Øx5a,Øx59,Øx5b,Øx58,Øxcb);}
[DB6]   37: #define INTREG rbp,rdi,rsi,rds,res,rdx,\
[LG4]   38:          rcx,rbx,rax,rip,rcs,flags
[2TØ]   39: #else
[RE6]   4Ø: #define INTREG res,rds,rdi,rsi,rbp,rsp\
[H25]   41:        ,rbx,rdx,rcx,rax,rip,rcs,flags
[UL4]   42: #define PUSHF   _asm{pushf}
[574]   43: #define POPF    _asm{popf}
[ØØØ]   44:
[4P2]   45: /* from conio.h */
[ØØØ]   46:
[1D3]   47: #define inportb   inp
[8U3]   48: #define outportb  outp
[ØØØ]   49:
[RE2]   5Ø: /*  from dos.h  */
[ØØØ]   51:
[554]   52: #define disable  _disable
[TU3]   53: #define enable   _enable
[HY4]   54: #define getvect _dos_getvect
[DØ5]   55: #define setvect _dos_setvect
[PY3]   56: #define keep    _dos_keep
[1Y3]   57: #define read    _dos_read
[6D4]   58: #define write   _dos_write
[H14]   59: #define open    _dos_open
[L94]   6Ø: #define close   _dos_close
[ØA4]   61: #define creat   _dos_creat
[NZØ]   62: #endif
[ØØØ]   63:
[NZØ]   64: #endif
```

Listing 1. »compatib.c« ist die Kompatibilitätsbox zwischen Turbo C und Quick C

```
[4W2]    1: /*Datei: hardint.c
[4X5]    2:   Funktion: Funktionsbibliothek für
[Z64]    3:             8259 Interruptcontroller
[MP1]    4:   Sprache: C
[3L4]    5:   Autor: Matthias Oberdorfer
[T15]    6:   (c)1991 DMV Widuch GmbH & Co. KG */
[ØØØ]    7:
[1P2]    8: #ifndef HARDINT
[7P2]    9: #define HARDINT
[ØØØ]   1Ø:
[PA3]   11: #include"compatib.c"
[ØØØ]   12:
[3X5]   13: /*  PC XT AT definitons port 2Ø hex  */
[ØØØ]   14:
[7W3]   15: #define TIMER           ØxØØØ1
[274]   16: #define KEYBOARD        ØxØØØ2
[NS3]   17: #define COM2            ØxØØØ8
[YR3]   18: #define COM1            ØxØØ1Ø
[L14]   19: #define FLOPPY          ØxØØ4Ø
[XT3]   2Ø: #define LPT1            ØxØØ8Ø
[Z46]   21: #define HARDDISK (is_AT()?Øx4ØØØ:ØxØØ2Ø)
[ØØØ]   22:
[5Y4]   23: /*  AT definitions port 2Ø hex  */
[ØØØ]   24:
[A24]   25: #define CASCADE         ØxØØØ4
[KT3]   26: #define LPT2            ØxØØ2Ø
[ØØØ]   27:
[3Z4]   28: /*  AT definitions port AØ hex  */
[ØØØ]   29:
[3N4]   3Ø: #define REALTIMECLOCK   Øx1ØØ
[M54]   31: #define MATHCOP         Øx2ØØØ
[ØØØ]   32:
[PG3]   33: /*  General defines  */
[ØØØ]   34:
[P93]   35: #define ALL     Øxffff
[773]   36: #define DEFAULT -1L
[733]   37: #define INT1_Ø  Øx2Ø
[B33]   38: #define INT1_1  Øx21
[453]   39: #define INT2_Ø  ØxAØ
[853]   4Ø: #define INT2_1  ØxA1
[ØØØ]   41:
[YV2]   42: /*  EOI-Commands  */
```

```
[ØØØ]   43:
[CT5]   44: #define EOI_NS      Øx2Ø/* n. spez. EOI */
[9D5]   45: #define EOI_S       Øx6Ø/* spez. EOI */
[XR6]   46: #define ROT_NS_EOI Øx9Ø/* rot. n.NS_EOI */
[BZ6]   47: #define ROT_A_S   Øx8Ø/*rot.auto.EOI(set)*/
[GW6]   48: #define ROT_A_C   ØxØØ/*rot.auto.EOI(clr)*/
[2K6]   49: #define ROT_S_EOI ØxeØ/*rot.n.spez.EOI*/
[LW6]   5Ø: #define SET_PRIO    ØxcØ/* set Priorität */
[YW5]   51: #define READ_COM    Øx4Ø/* Lesebefehl */
[ØØØ]   52:
[M83]   53: /*  Register Token  */
[ØØØ]   54:
[4X2]   55: #define IRR     ØxØØ
[8X2]   56: #define ISR     Øx01
[RW2]   57: #define IMR     Øx02
[ØØØ]   58:
[9V6]   59: /*  is_AT() :stellt fest ob Rechner AT ist
[ØØØ]   6Ø:
[5H3]   61:     Rückgabe    Bedeutung
[492]   62:     ---------------------
[4L2]   63:        Ø        PC oder XT
[PD2]   64:        1        AT              */
[ØØØ]   65:
[B96]   66: #define is_AT() (*((unsigned char far *)\
[683]   67:                  ØxfØ00fffe)==Øxfc?1:Ø)
[ØØØ]   68:
[KR5]   69: /*  EOI_not_spec(): Abschluß eines
[DE3]   7Ø:                     Hardwareinterrupts
[YV4]   71:     EOI_spec()    : Abschluß eines
[6F5]   72:             speziellen Hardwareinterrupts*/
[ØØØ]   73:
[3Z6]   74: #define EOI_not_spec()  outportb(INT1_Ø,\
[L91]   75:                          EOI_NS)
[BF6]   76: #define EOI_spec()      outportb(INT1_Ø,\
[VØ1]   77:                          EOI_S)
[ØØØ]   78:
[HR3]   79: /*  ANSI-Prototypes  */
[ØØØ]   8Ø:
[T55]   81: void permit_int(unsigned nr);
[D35]   82: void forbid_int(unsigned nr);
[UZ6]   83: void init_cont(int adr1,int adr2,long nr);
[LN4]   84: void set_prio(unsigned nr);
[HE5]   85: unsigned read_reg(unsigned tok);
[ØØØ]   86:
[696]   87: /*  permit_int(nr): erlaubt Hardeware-
[6F5]   88:     Interrupt nr es können mehrere
[VT5]   89:     Interrupt gleichzeitig zugelassen
[X31]   9Ø:     werden
[Ø36]   91:     nr : TIMER,KEYBOARD,COM2,COM1,FLOPPY,
[G72]   92:          LPT1,HARDDISK
[L64]   93:     nr (nur AT) : CASCADE,LPT2,
[KN3]   94:          REALTIMECLOCK,MATHCOP
[B61]   95:     nr : ALL
[XP1]   96:     Beispiele:
[BD4]   97:     TIMER und FLOPPY zulassen
[YM4]   98:     permit_int(TIMER¦FLOPPY);
[754]   99:     Alle Interrupt zulassen
[2T3]   1ØØ:      permit_int(ALL);          */
[ØØØ]   1Ø1:
[5F3]   1Ø2: void permit_int(nr)
[652]   1Ø3: unsigned nr;
[WD1]   1Ø4: { PUSHF;
[RM1]   1Ø5:   disable();
[B55]   1Ø6:   /* 1. Interrupt controller ein */
[WL5]   1Ø7:   outportb(INT1_1,inportb(INT1_1)&
[9U2]   1Ø8:                   ((nr&Øxff)^Øxff));
[C55]   1Ø9:   /* 2. Interrupt controller ein */
[RP1]   11Ø:   if(is_AT())
[TK5]   111:   outportb(INT2_1,inportb(INT2_1)
[5B2]   112:                   &((nr>>8)^Øxff));
[9WØ]   113:   POPF;
[U7Ø]   114: }
[ØØØ]   115:
[7K6]   116: /*  forbid_int(nr): verbietet Hardeware-
[6F5]   117:       Interrupt nr es können mehrere
[HL6]   118:       Interrupts gleichzeitig unterdrückt
[N91]   119:       werden.
[Ø36]   12Ø:    nr : TIMER,KEYBOARD,COM2,COM1,FLOPPY,
[G72]   121:         LPT1,HARDDISK
[L64]   122:    nr (nur AT) : CASCADE,LPT2,
[KN3]   123:         REALTIMECLOCK,MATHCOP
[B61]   124:    nr : ALL
[XP1]   125:    Beispiele:
[ØR4]   126:    TIMER und FLOPPY verbieten:
[DL4]   127:     forbid_int(TIMER¦FLOPPY);
[CR4]   128:    Alle Interrupts verbieten:
[2T3]   129:     forbid_int(ALL);          */
[ØØØ]   13Ø:
[TC3]   131: void forbid_int(nr)
[652]   132: unsigned nr;
[WD1]   133: { PUSHF;
[RM1]   134:   disable();
[E65]   135:   /* 1. Interrupt controller aus */
[PK5]   136:   outportb(INT1_1,inportb(INT1_1)
[2V1]   137:                   ¦(nr&Øxff));
[815]   138:   /* 2.Interrupt controller aus */
[RP1]   139:   if(is_AT())
[TK5]   14Ø:   outportb(INT2_1,inportb(INT2_1)
[WH1]   141:                   ¦(nr>>8));
```

```
[9WØ]   142:   POPF;
[U7Ø]   143: }
[ØØØ]   144:
[CA4]   145: /*  init_cont(adr1, adr2, nr)
[T54]   146:     intitialisiert die beiden
[ME4]   147:     Interruptcontroller 8259A
[H86]   148:     adr  : zeigen wo die Interruptvekoten
[852]   149:            liegen sollen
[3Ø6]   15Ø:     adr1 : Adresse der Interruptvektoren
[XS4]   151:            des 1.Interruptcontrollers
[2B2]   152:            (DEFAULT : Øx08)
[4Ø6]   153:     adr2 : Adresse der Interruptvektoren
[ZS4]   154:            des 2.Interruptcontrollers
[RA2]   155:            (DEFAULT : Øx7Ø)
[8X4]   156:     nr   : erlaubte interrupt nach
[TU2]   157:            Initialisierung
[283]   158:           (siehe permit_int())
[TG1]   159:     Beispiel:
[PE6]   16Ø:    init_cont(DEFAULT, DEFAULT, DEFAULT); */
[ØØØ]   161:
[ZS4]   162: void init_cont(adr1,adr2,nr)
[5D2]   163: int  adr1,adr2;
[HD1]   164: long nr;
[HF5]   165: { unsigned old_nr=read_reg(IMR);
[G31]   166:   PUSHF;
[RM1]   167:   disable();
[AL4]   168:   /* 1. Interrupt Controller */
[7X5]   169:   outportb(INT1_Ø,(is_AT()?Øx11:Øx13));
[YT5]   17Ø:   outportb(INT1_1,(adr1>Ø?adr1:Øx08));
[RP1]   171:   if(is_AT())
[8R3]   172:   outportb(INT1_1,ØxØ4);
[ØZ5]   173:   outportb(INT1_1,(is_AT()?Øxød:Øx09));
[BL4]   174:   /* 2. Interrupt Controller */
[RP1]   175:   if(is_AT())
[LR3]   176:   { outportb(INT2_Ø,Øx11);
[UT5]   177:     outportb(INT2_1,(adr2>Ø?adr2:Øx7Ø));
[8R3]   178:     outportb(INT2_1,Øx02);
[FR3]   179:     outportb(INT2_1,Øx09);
[U7Ø]   18Ø:   }
[9WØ]   181:   POPF;
[UR2]   182:   forbid_int(ALL);
[ZU4]   183:   permit_int((nr>ØL?(unsigned)
[FX4]   184:            (nr&ØxffffL):old_nr^Øxffff));
[U7Ø]   185: }
[ØØØ]   186:
[ØC6]   187: /*  set_prio(nr) : setzt Priorität der
[TØ6]   188:     Interruptbehandlung normalerweise
[9D4]   189:     hat TIMER höchste und LPT1
[BT3]   19Ø:     niedrigste Priorität
[TT3]   191:     nr : eines der Geräte :
[8D5]   192:          TIMER,KEYBOARD,CASCADE,COM2,COM1,
[WY2]   193:          LPT2,FLOPPY,LPT1
[GM4]   194:     Beispiel: set_prio(COM2);   */
[ØØØ]   195:
[LY2]   196: void set_prio(nr)
[652]   197: unsigned nr;
[SF2]   198: { unsigned i=Ø;
[UU5]   199:   /* Feststellen der Interrupt-ebene */
[M13]   2ØØ:   while(!(nr&Øx01)&&i<8)
[SK2]   2Ø1:   { nr = nr>>1; i++;
[U7Ø]   2Ø2:   }
[N46]   2Ø3:   if(i<8) outportb(INT1_Ø,SET_PRIO|i);
[U7Ø]   2Ø4: }
[ØØØ]   2Ø5:
[Z55]   2Ø6: /*  read_reg(tok) : liest die drei
[SX5]   2Ø7:       Interrupt-Controller-Register aus
[DT6]   2Ø8:     tok: spezifiziert zulesendes Register:
[N32]   2Ø9:          IMR, IRR, ISR
[4L4]   21Ø:     Dabei wird folgendes Format
[F33]   211:     zuückgeliefert:
[DN6]   212:     unteres Byte : 1. Interrupt-Controller
[LC6]   213:     oberes  Byte : 2. Interrupt-Controller
[TG1]   214:     Beispiel:
[GW3]   215:     if(read_reg(IMR)&TIMER)
[294]   216:      printf("TIMER aktiv\n");  */
[ØØØ]   217:
[CV3]   218: unsigned read_reg(tok)
[K92]   219: unsigned tok;
[BT2]   22Ø: { unsigned ret=Ø;
[G31]   221:   PUSHF;
[RM1]   222:   disable();
[SW1]   223:   switch(tok)
[3V1]   224:   { case IRR:
[7G1]   225:     case ISR:
[CL4]   226:       outportb(INT1_Ø,Øx0a|tok);
[5G3]   227:       ret=inportb(INT1_Ø);
[RP1]   228:       if(is_AT())
[4R4]   229:       { outportb(INT2_Ø,Øx0a|tok);
[9C4]   23Ø:         ret ¦=(inportb(INT2_Ø)<<8);
[U7Ø]   231:       }
[PØ1]   232:     break;
[FF1]   233:     case IMR:
[8G3]   234:       ret=inportb(INT1_1);
[RP1]   235:       if(is_AT())
[BC4]   236:         ret ¦=(inportb(INT2_1)<<8);
[PØ1]   237:     break;
```

Listing 2. »hardint.c« ist die Funktionsbibliothek für den Interruptcontroller

```
[U7Ø]   238:    }
[9WØ]   239:    POPF;
[XZ1]   24Ø:    return(ret);
[U7Ø]   241: }
[ØØØ]   242:
[NZØ]   243: #endif
```

Listing 2. »hardint.c« ist die Funktionsbibliothek für den Interruptcontroller

```
[UP2]     1: /*Datei: serial.c
[MØ6]     2:   Funktion: Kommunikation zweier PCs
[GW3]     3:             und Datenübertragung
[MP1]     4:   Sprache: C
[3L4]     5:   Autor: Matthias Oberdorfer
[T15]     6:   (c)1991 DMV Widuch GmbH & Co. KG */
[ØØØ]     7:
[2C2]     8: #ifndef SERIAL
[8C2]     9: #define SERIAL
[7M2]    1Ø: #include<dos.h>
[P43]    11: #include"hardint.c"
[ØØØ]    12:
[LL4]    13: typedef unsigned int word;
[ØØØ]    14:
[9Z4]    15: /* Basisadresse für serielle
[9Ø4]    16:    Kommunikationsports */
[ØØØ]    17:
[9C6]    18: #define COM1_BASE Øx3ØØ /* COM1:Portadr */
[CC6]    19: #define COM2_BASE Øx2ØØ /* COM2:Portadr */
[ØØØ]    2Ø:
[MD4]    21: /* Adressen der asychronen
[815]    22:    Kommunikationsports beim PC
[FT4]    23:    relativ zur Basisadresse */
[ØØØ]    24:
[884]    25: /* TX-buffer                  DLAB=Ø*/
[KG4]    26: #define TXB(adr) (ØxF8|adr)
[684]    27: /* RX-buffer                  DLAB=Ø*/
[EG4]    28: #define RXB(adr) (ØxF8|adr)
[YY5]    29: /* Interrupt Enable Register DLAB=Ø*/
[6G4]    3Ø: #define IER(adr) (ØxF9|adr)
[SØ5]    31: /* Divisor Latch Low          DLAB=1*/
[PF4]    32: #define DLL(adr) (ØxF8|adr)
[M15]    33: /* Divisor Latch High         DLAB=1*/
[9F4]    34: #define DLH(adr) (ØxF9|adr)
[GH6]    35: /* Interrupt Identification Register */
[AH4]    36: #define IIR(adr) (ØxFA|adr)
[KC4]    37: /* Line Control Register */
[1H4]    38: #define LCR(adr) (ØxFB|adr)
[TD4]    39: /* Modem Control Register */
[6H4]    4Ø: #define MCR(adr) (ØxFC|adr)
[X24]    41: /* Line Status Register */
[RK4]    42: #define LSR(adr) (ØxFD|adr)
[BD4]    43: /* Modem Status Register */
[WK4]    44: #define MSR(adr) (ØxFE|adr)
[ØØØ]    45:
[ØØØ]    46:
[S25]    47: /* Interrupt Enable Register */
[ØØØ]    48:
[2K6]    49: #define DATA_REC  1/*Daten in RBX-Puffer*/
[1G6]    5Ø: #define THR_EMPTY 2/*THR-Puffer leer */
[UX6]    51: #define LINE_CHG  4/*Leit.zustandswechsel
[DK3]    52:                         erzeugt Interrupt*/
[647]    53: #define MODEM_CHG 8/*Modemzustandswechsel
[DK3]    54:                         erzeugt Interrupt*/
[ØØØ]    55:
[GH6]    56: /* Interrupt Identification Register */
[ØØØ]    57:
[KN6]    58: #define I_DATA_REC  4/*Daten in RBX-Puf*/
[PR6]    59: #define I_THR_EMPTY 2/*THR-Puffer leer */
[GS6]    6Ø: #define I_LINE_CHG  6/*LZW -> Interrupt */
[4V6]    61: #define I_MODEM_CHG Ø/*MZW -> Interrupt */
[ØØØ]    62:
[X24]    63: /* Line Status Register */
[ØØØ]    64:
[PR6]    65: #define DATA_READY 1 /*Daten angekommen */
[HH6]    66: #define OVERFLOW   2 /*RBX übergelaufer */
[WR5]    67: #define PARITY     4 /*Parityfehler */
[5F5]    68: #define FRAME      8 /*Rahmenfehler */
[HV5]    69: #define BREAK     16 /*Break empfangen */
[B35]    7Ø: #define THR_LEER  32 /*THR leer */
[G55]    71: #define TSR_LEER  64 /*TSR leer */
[ØØØ]    72:
[TD4]    73: /* Modem Control Register */
[ØØØ]    74:
[M36]    75: #define DTR     1  /*Data Terminal Ready*/
[Ø75]    76: #define RTS     2  /*Ready To Send */
[EK4]    77: #define OUT1    4  /*Not used */
[EL4]    78: #define OUT2    8  /*Interrupt
[3C4]    79:                 von 825Ø zum 8259 schalten */
[265]    8Ø: #define LOOP    16 /*Selbsttest */
[ØØØ]    81:
[BD4]    82: /* Modem Status Register */
[ØØØ]    83:
[A86]    84: #define DCTS    1/*Detect Clear To Send */
[E46]    85: #define DDSR    2/*Detect Data Set Ready*/
[ER4]    86: #define DRI     4/*Detect Ring */
[5E6]    87: #define DRLSD   8/*Detect Trägersignal */
[Z75]    88: #define CTS    16/*Clear To Send */
[ØB5]    89: #define DSR    32/*Data Set Ready */
[TG3]    9Ø: #define RI     64/*Ring */
```

```
[AG5]    91: #define RLSD  128/*Trägersignal */
[ØØØ]    92:
[SH6]    93: /* dlab(nr, c): setzt oder löscht DLAB-Bit
[XX2]    94: ** im LCR-Rgister */
[ØØØ]    95:
[YG6]    96: #define dlab(adr,c){ outportb(LCR(adr),c\
[584]    97:         ?inportb(LCR(adr))|Øx8Ø:\
[Ø34]    98:         inportb(LCR(adr))&Øx7f);}
[ØØØ]    99:
[WX1]   1ØØ: /* Parität */
[ØØØ]   1Ø1:
[213]   1Ø2: #define NONE     ØxØØ
[6Y2]   1Ø3: #define ODD      Øx Ø8
[VØ3]   1Ø4: #define EVEN     Øx18
[ØØØ]   1Ø5:
[8M2]   1Ø6: typedef struct
[TZ4]   1Ø7: { word p_adr; /* Portadresse */
[NR6]   1Ø8:   word int_nr; /* Soft-Interruptnummer */
[B67]   1Ø9:   word hard_int; /*Interruptcontrollerbit*/
[HH5]   11Ø:   void (interrupt far *int_vect)();
[TT5]   111:   void (interrupt far *save_vect)();
[PF5]   112:   word save_ier; /* IER gesichert */
[KG5]   113:   word save_mcr; /* MCR gesichert */
[6G5]   114:   word save_imr; /* IMR gesichert */
[ØØ1]   115: } com;
[ØØØ]   116:
[KF6]   117: void set_para(word adr,unsigned long \
[NM6]   118: baud,word datelen,word stopbits,word \
[6C1]   119: parity);
[ØØØ]   12Ø:
[ØN6]   121: /* set_para(adr, baud, datelen, stopbits\
[TD1]   122:                  ,parity)
[AA5]   123:    setzt Schnittstellenparameter
[9Y4]   124:    adr     : Basisadresse des Ports
[YØ3]   125:    baud    : Ø-1152ØØ Baud
[EN4]   126:    datelen : 5-8  Bit-Paketlänge
[L22]   127:    stopbits: Ø-1
[264]   128:    parity  : NONE, EVEN, ODD */
[ØØØ]   129:
[9S6]   13Ø: void set_para(adr,baud,datelen,stopbits
[TD1]   131:                 ,parity)
[ØA3]   132: unsigned long baud;
[2R5]   133: word datelen,stopbits,adr,parity;
[SC3]   134: { /* Setze Baudrate */
[HR1]   135:   dlab(adr,1);
[ØG5]   136:   outportb(DLL(adr),((unsigned int)
[FZ2]   137:                (1152ØØL/baud))&Øxff);
[WF5]   138:   outportb(DLH(adr),((unsigned int)
[1K2]   139:                (1152ØØL/baud))>>8);
[HR6]   14Ø:   /* setze Datenpaketlänge und Stopbits */
[ER1]   141:   dlab(adr,Ø);
[1F4]   142:   outportb(LCR(adr),parity|
[H25]   143:          ((datelen>>1)-1)|(stopbits?4:Ø));
[U7Ø]   144: }
[ØØØ]   145:
[6K6]   146: /* init_com(COM, baud, datelen, stopbits,
[XH2]   147:                              parity,ier,mcr)
[1B6]   148:    initialisiert serielle Schnittstelle
[K55]   149:    COM     : Strutur mit Portinfos
[YØ3]   15Ø:    baud    : Ø-1152ØØ Baud
[EN4]   151:    datelen : 5-8  Bit-Paketlänge
[L22]   152:    stopbits: Ø-1
[BW3]   153:    parity  : NONE, EVEN, ODD
[L76]   154:    ier     : DATA_REC,THR_EMPTY,LINE_CHG,
[NL1]   155:              MODEM_CHG
[195]   156:    mcr     : DTR,RTS,OUT1,OUT2,LOOP */
[ØØØ]   157:
[RG6]   158: void (interrupt far *init_com(COM,baud
[TX5]   159:    ,datelen,stopbits,parity,ier,mcr))()
[8D6]   16Ø: word datelen,stopbits,parity,ier,mcr;
[ØA3]   161: unsigned long baud;
[3F1]   162: com *COM;
[7V5]   163: { void (interrupt far *old_vect)();
[K24]   164:   /* Sichern von Variablen */
[GK6]   165:   COM->save_mcr =inportb(MCR(COM->p_adr));
[2K6]   166:   COM->save_ier =inportb(IER(COM->p_adr));
[SY4]   167:   COM->save_imr =read_reg(IMR);
[Z96]   168:   COM->save_vect=getvect(COM->int_nr);
[B15]   169:   /* Interruptvector verbiegen */
[S65]   17Ø:   old_vect=getvect(COM->int_nr);
[PC6]   171:   setvect(COM->int_nr,COM->int_vect);
[YD3]   172:   /* Parameter setzen */
[AN5]   173:   set_para(COM->p_adr,baud,datelen,
[Z23]   174:                      stopbits,parity);
[6B5]   175:   /* 825Ø Register initialisieren */
[HT4]   176:   /* Löschen von RXB, IIR, MSR */
[NB4]   177:   inportb(RXB(COM->p_adr));
[PB4]   178:   inportb(IIR(COM->p_adr));
[WC4]   179:   inportb(MSR(COM->p_adr));
[2X4]   18Ø:   while(inportb(LSR(COM->p_adr))
[Y12]   181:                      & DATA_READY)
[NB4]   182:   inportb(RXB(COM->p_adr));
[455]   183:   outportb(IER(COM->p_adr),ier);
[C55]   184:   outportb(MCR(COM->p_adr),mcr);
[SL5]   185:   /* 8259 Interruptcontroller ein */
[GK4]   186:   permit_int(COM->hard_int);
[DØ3]   187:   return(old_vect);
[U7Ø]   188: }
[ØØØ]   189:
[E82]   19Ø: exit_com(COM)
```

```
[3F1]  191: com *COM;
[SY6]  192: { outportb(MCR(COM->p_adr),COM->save_mcr);
[KZ6]  193:   outportb(IER(COM->p_adr),COM->save_ier);
[UR2]  194:   forbid_int(ALL);
[NV5]  195:   permit_int(COM->save_imr^Øxffff);
[1G6]  196:   setvect(COM->int_nr,COM->save_vect);
[U7Ø]  197: }
[ØØØ]  198:
[NZØ]  199: #endif
```

Listing 3. »serial.c« unterstützt die Kommunikation zweier PCs

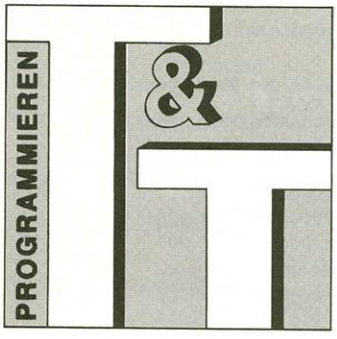
```
[7S3]    1: /*Programm: communic.c
[P66]    2:   Funktion: einfaches Datenaustausch-
[1G1]    3:             Programm
[MP1]    4:   Sprache: C
[3L4]    5:   Autor: Matthias Oberdorfer
[T15]    6:   (c)1991 DMV Widuch GmbH & Co. KG */
[ØØØ]    7:
[AY2]    8: #include"serial.c"
[1C4]    9: /* Anwendung von Serial.c */
[ØØØ]   1Ø:
[GS2]   11: #include"queue.c"
[TZ2]   12: #define BUFLEN  1ØØØ
[B43]   13: char buff[BUFLEN];
[NE3]   14: struct queue Queue;
[ØØØ]   15:
[L92]   16: com *COM_PTR;
[ØØØ]   17:
[C96]   18: word txbfree=1;      /*Sendepuffer frei*/
[XW6]   19: int far push_ch(); /*Forward-Delaration*/
[ØØØ]   2Ø:
[X36]   21: void (interrupt far *old_ser)(void);
[LF5]   22: void interrupt far new_ser(void)
[7P5]   23: { forbid_int(COM_PTR->hard_int);
[ØB1]   24:   enable();
[SØ6]   25:   switch(inportb(IIR(COM_PTR->p_adr)))
[4W3]   26:   { /* Zeichen empfangen */
[FU2]   27:     case I_DATA_REC:
[ØZ3]   28:       dlab(COM_PTR->p_adr,Ø);
[3S2]   29:       push_ch(&Queue,
[5A5]   3Ø:       inportb(RXB(COM_PTR->p_adr)));
[PØ1]   31:     break;
[3B3]   32:     /* Sendepuffer leer */
[L53]   33:     case I_THR_EMPTY:
[NP1]   34:       txbfree=1;
[PØ1]   35:     break;
[U7Ø]   36:   }
[RM1]   37:   disable();
[PP2]   38:   EOI_not_spec();
[UD5]   39:   permit_int(COM_PTR->hard_int);
[U7Ø]   4Ø: }
[ØØØ]   41:
[ØØØ]   42:
[6E3]   43: int sendchar(ch,COM)
[P91]   44: int ch;
[3F1]   45: com *COM;
[SR2]   46: { unsigned wait;
[S15]   47:   /* warten auf Sendepuffer leer */
[F76]   48:   for(wait=3ØØØØ; !txbfree&&wait; wait--)
[BZ5]   49:     if(inportb(LSR(COM->p_adr))&THR_LEER)
[NP1]   5Ø:   txbfree=1;
[863]   51:   if(!wait) return(-2);
[P84]   52:   /* und senden des Zeichens */
[LP1]   53:   txbfree=Ø;
[525]   54:   outportb(TXB(COM->p_adr),ch);
[124]   55:   return(Ø); /* kein Fehler */
[U7Ø]   56: }
[ØØØ]   57:
[UWØ]   58: main()
[HK1]   59: { com COM;
[SG1]   6Ø:   int ch =Ø;
[HX2]   61:   unsigned ready=Ø;
[RS2]   62:   unsigned count;
[8V3]   63:   COM.p_adr   = COM1_BASE;
[T13]   64:   COM.int_nr  = Øx Øc;
[793]   65:   COM.hard_int= COM1;
[3Z3]   66:   COM.int_vect= new_ser;
[P95]   67:   init_queue(&Queue,buff,BUFLEN);
[4V6]   68:   printf("\n\n Serielle Datenübertragung"
[NXØ]   69:          "\n");
[DG2]   7Ø:   printf(" \n\n");
[LT5]   71:   printf(" Taste        Bedeutung\n\n");
[T14]   72:   printf(" s    send data\n");
[RE4]   73:   printf(" p    print data\n");
[LH4]   74:   printf(" c    command.com\n");
[NW4]   75:   printf(" r    show register\n");
[1D5]   76:   printf(" i    interactiv send\n");
[ET4]   77:   printf(" e    exit program\n");
[ØM2]   78:   /* für new_ser */
[GH2]   79:   COM_PTR = &COM;
[X56]   8Ø:   old_ser=init_com(&COM,1152ØØL,8,Ø,NONE,
[FF5]   81:                DATA_REC|THR_EMPTY,DTR|OUT2);
[PY1]   82:   while(!ready)
[HD2]   83:   switch(getch())
[6N1]   84:   { case 'e':
[391]   85:       ready=1;
[PØ1]   86:     break;
[P91]   87:     case 'c':
```

```
[ER3]   88:       system("command.com");
[PØ1]   89:     break;
[KB1]   9Ø:     case 'r':
[HK5]   91:       printf("Inhalte der Register\n");
[7Z2]   92:       printf("mcr %x\n",
[A74]   93:              inportb(MCR(COM.p_adr)));
[5Z2]   94:       printf("ier %x\n",
[874]   95:              inportb(IER(COM.p_adr)));
[1E5]   96:       printf("imr %x\n",read_reg(IMR));
[RØ3]   97:     printf("msr %x\n",
[U84]   98:            inportb(MSR(COM.p_adr)));
[PØ1]   99:     break;
[PB1]  1ØØ:     case 's':
[H64]  1Ø1:       printf("Geben Sie das zu \
[ØM5]  1Ø2:        sendende Zeichen ein (ASC-Wert) :");
[L32]  1Ø3:       scanf("%d",&ch);
[394]  1Ø4:       printf("Wie oft soll es \
[H63]  1Ø5:         gesendet werden :");
[UU2]  1Ø6:       scanf("%d",&count);
[G85]  1Ø7:       while(count--) sendchar(ch,&COM);
[PØ1]  1Ø8:     break;
[AB1]  1Ø9:     case 'p':
[TX4]  11Ø:       printf("Empfangene Zeichen : ");
[5L4]  111:       while((ch=pop_ch(&Queue))>=Ø)
[8A2]  112:         printf("%c",ch);
[Z53]  113:       printf("\nEnde der \
[524]  114:        empfangenen Zeichen\n");
[PØ1]  115:     break;
[EA1]  116:     case 'i':
[RL6]  117:     printf("q to stop interactive send\n");
[NBØ]  118:       do
[TV2]  119:       { while(!kbhit());
[M54]  12Ø:         sendchar((ch=getch()),&COM);
[8A2]  121:         printf("%c",ch);
[PD2]  122:       }while(ch!='q');
[DØ2]  123:       printf("\n");
[PØ1]  124:     break;
[U7Ø]  125:   }
[SH2]  126:   exit_com(&COM);
[PP2]  127:   printf("Ende\n");
[U7Ø]  128: }
```

Listing 4. »communic.c« demonstriert den Datenaustausch zwischen zwei PCs

```
[2K2]    1: /*Datei: queue.c
[HL6]    2:   Funktion: Verwaltung für FIFO-Puffer
[MP1]    3:   Sprache: C
[3L4]    4:   Autor: Matthias Oberdorfer
[T15]    5:   (c)1991 DMV Widuch GmbH & Co. KG */
[ØØØ]    6:
[A52]    7: #ifndef QUEUE
[G52]    8: #define QUEUE
[ØØØ]    9:
[452]   1Ø: struct queue
[844]   11: { /* Ringpuffer struct */
[3T5]   12:   char *   q_buf; /* Zeiger auf Puffer */
[4B6]   13:   unsigned q_len; /* Länge des Puffer */
[B56]   14:   unsigned q_write; /* Schreibzeiger */
[NC5]   15:   unsigned q_read; /* Lesezeiger */
[BDØ]   16: };
[ØØØ]   17:
[247]   18: #define init_queue(q,ptr,len){ (q)->q_buf\
[3K3]   19:         =ptr; (q)->q_len=len;\
[GB5]   2Ø:         (q)->q_write=Ø; (q)->q_read=Ø;}
[ØØØ]   21:
[GY5]   22: /* push_ch(ch) fügt ch in puffer ein */
[ØØØ]   23:
[HL3]   24: int far push_ch(q,ch)
[1R2]   25: struct queue *q;
[P91]   26: int ch;
[TS4]   27: { q->q_buf[q->q_write]=ch;
[G94]   28:   if((q->q_write+1<q->q_len?
[ØC4]   29:    q->q_write+1:Ø)!=q->q_read)
[PP4]   3Ø:   { q->q_write = (q->q_write+1
[ND4]   31:     <q->q_len?q->q_write+1:Ø);
[TL1]   32:     return(Ø);
[U7Ø]   33:   }
[WT1]   34:   return(-1);
[U7Ø]   35: }
[ØØØ]   36:
[853]   37: int far push_ch();
[ØØØ]   38:
[145]   39: /* pop_ch() : holt ch aus Puffer */
[ØØØ]   4Ø:
[GV2]   41: int far pop_ch(q)
[1R2]   42: struct queue *q;
[C44]   43: { if(q->q_read>=q->q_len)
[NX1]   44:   q->q_read=Ø;
[YØ4]   45:   if(q->q_write!=q->q_read)
[MY4]   46:   return(q->q_buf[q->q_read++]);
[WT1]   47:   return(-1);
[U7Ø]   48: }
[CV2]   49: int far pop_ch();
[ØØØ]   5Ø:
[NZØ]   51: #endif
```

Listing 5. »queue.c« verwaltet den FIFO-Puffer für den Datenaustausch



»Tips & Tricks für Programmierer« ist eine Rubrik zum Mitmachen. Sie müssen kein Schriftsteller sein, damit ein Beitrag aus Ihrer Feder (beziehungsweise aus Ihrer Tastatur) auf diesen Seiten veröffentlicht wird. Einschicken können Sie alles, was irgendwie mit der Programmierung des PC zusammenhängt. Das muß keine seitenlange Superroutine sein, auch eine nützliche Assembler-Routine, eine genial programmierte C-Funktion oder ein Patch für MS-DOS hat gute Chancen, in »Tips & Tricks für Programmierer« abgedruckt zu werden.
Und ein angemessenes Honorar gibt es für Ihre Bemühungen natürlich auch. Es richtet sich nach der Qualität der Tricks und der Dokumentation. Pauschalhonorare gibt es bei uns nicht, schließlich möchten wir gute Leistungen entsprechend würdigen.
Bitte schicken Sie eine Diskette mit einer kurzen Programmbeschreibung und dem Quelltext an:

DMV-Redaktionsbüro, DOS International, Stichwort »Tips & Tricks«,
Gruberstraße 46c, 8011 Poing bei München

Wir freuen uns über jede Zuschrift. (et)

Tips und Tricks für Programmierer

Mit eigener Software kontrollieren Sie die Hardware. In mathematischen Programmen berechnen Sie »Pi« und »E« auf zehn Nachkommastellen sowie Funktionsterme Ihrer Wahl. Sie erhalten Hintergrundwissen zu dBase-Dateien, römischen Zahlen, Kopier- und Virenschutzmethoden. In OOP programmieren Sie eine Virensimulation.

Kontrolle über Tastatur, Cursor und Maus

In manchen Anwendungen scheint es wünschenswert, Routinen zur Kontrolle von Tastatur, Cursor, Maus und Bildschirm zu erhalten. In der Unit »utility.pas« (Listing 1) finden Sie diese Anforderungen in einem Werkzeug vereint (Bild 1). Im Implementationsteil, ab Zeile 45, finden Sie in den Dokumentarzeilen aufgeführt, wie Utility Tastaturpuffer, Status-, Cursor- und Maustasten bearbeitet. Im folgenden finden Sie eine erklärende Kurzbeschreibung der einzelnen Routinen.

- Routinen zur Kontrolle der Tastatur:

```
procedure ClrKeyBuf;
```

(Zeile 51) löscht den Tastaturpuffer.

```
procedure GetStatusKeys;
```

(Zeile 55) schreibt den Zustand der Statustasten in folgende, im Interface-Teil deklarierte (globale) Variablen (Zeile 23):

```
var _Insert,_Shift,_Ctrl,_Alt,_NumLock,
_CapsLock,_ScrollLock: Boolean;
```

Jede dieser »Statusvariablen« beginnt also mit einem Unterstrich.

- Routinen zur Kontrolle des Cursors:

```
procedure HideCursor;
```

in Zeile 75 »versteckt« den Cursor durch einen BIOS-Aufruf.

```
procedure ShowCursor;
```

zeigt den Cursor wieder an, wobei die Unit im »Hauptprogramm«, die beim Programmstart in Hilfsvariablen gesicherten Werte für Anfangs- und End-Rasterzeilen wieder setzt.

- Routinen zur Kontrolle und Abfrage der Maus:

```
procedure ResetMouse;
```

(Zeile 85) initialisiert die Maus und setzt eine günstige Kombination von »cursor masks« und »screen masks«, wobei die Utility-Unit zwischen Hercules- und Farbgrafikkarten unterscheidet. Die »masks« beeinflussen die Verknüpfung des Mauscursors mit dem Texthintergrund und dessen Farbe. Je nachdem, ob eine Maus vorhanden ist, wird die globale Variable

```
var Mouse: Boolean;
```

(Zeile 29) auf »True« oder »False« gesetzt.

```
procedure ShowMouse;
```

in Zeile 103 zeigt den Mauscursor. Analog dazu finden Sie in Zeile 98

```
procedure HideMouse;
```

womit die Unit den Mauscursor wieder versteckt. Dieser Aufruf muß mit »ShowMouse« jeweils paarweise verwendet werden, um ordnungsgemäß arbeiten zu können.

```
function MouseKeys: Byte;
```

(Zeile 108) liefert den codierten Status der Maustasten zurück. Hier erhalten Sie den Wert 0, wenn keine Maus angeschlossen ist. Die Bedeutung der drei untersten Bits finden Sie in Tabelle 1.

```
procedure GetMouseXY(var x,y: Byte);
```

(Zeile 116) liefert die x- und y-Koordinate der Mausposition zurück. Hier rechnet Utility auf Textkoordinaten um, das heißt, Werte zwischen

```
1<=x<=80
1<=y<=25
```

liefern x = 1 und y = 1, wenn kein Maustreiber angeschlossen ist.

```
procedure PutMouseXY(x,y: Byte);
```

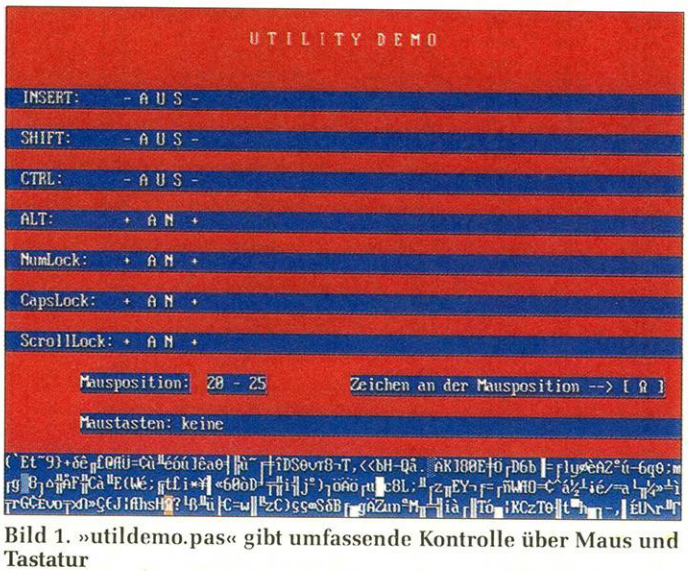


Bild 1. »utildemo.pas« gibt umfassende Kontrolle über Maus und Tastatur

Kombination	Verknüpfung	Wirkung
a	MouseKeys AND 1 = 1	links gedrückt
b	MouseKeys AND 4 = 4	Mitte gedrückt
c	MouseKeys AND 2 = 2	rechts gedrückt

Tabelle 1. Die Funktion »MouseKeys«, die den Status der Maustasten wiedergibt

(Zeile 128) setzt den Mauscursor an die Stelle »x,y« des Textbildschirms.

```
procedure SetMouseRange(MinX,MinY,MaxX,MaxY: Byte);
```

(ab Zeile 136) definiert einen Bereich, dessen Grenzen der Mauscursor nicht überschreiten kann.

```
procedure WaitMouse;
```

(Zeile 145) wartet solange, bis Sie alle Maustasten losgelassen haben. Diese Prozedur ist beispielsweise nützlich bei Realisierung eines Doppelklicks.

- Routinen zum direkten Zugriff auf den Bildschirm:

Diese Routinen wirken nur im Textmodus, also im Bildschirmformat von 80 Spalten mal 25 Zeilen.

```
procedure SetBlink(On: Boolean);
```

(Zeile 163) wählt einen »blinkenden Vordergrund« oder »hellen Hintergrund« für Textfarben, welche höhere Werte als 127 aufweisen. Dies gilt nur für EGA- und VGA-Grafikkarten. Für Hercules oder CGA müssen Sie die Routine aus der DOS International 11'90, Seite 36, einbinden. Korrekt muß dort die Zeile heißen:

```
ModReg: Byte Absolute $40: $65;
```

Für die vom übrigen Text unabhängige Bildschirmumrahmung, welche wiederum nur für EGA- und VGA-Grafikkarten gilt, finden Sie in Zeile 170

```
procedure Border(Col: Byte);
```

eine Prozedur, welche die Farbe des Bildrahmens setzt.

```
procedure SetColor(Att: Byte);
```

in Zeile 175 setzt die Farbe, in der Text ausgegeben wird. Diese Prozedur faßt die Wirkung von »TextColor« und »TextBackground« aus der Unit »Crt« zusammen. Dazu muß allerdings diese Unit im Hauptprogramm eingebunden sein.

```
procedure WriteChar(x,y: Byte; CH: Char);
```

(Zeile 179) demonstriert die Benutzung der Zeigervariablen »Screen«, die den Bildschirmspeicher nachbildet und somit eine direkte Manipulation des Bildschirmspeichers gestattet. Hierbei bestehen mit

```
var Att:Byte; CH:Char:
```

die Kombinationen, welche Sie Tabelle 2 entnehmen. Dabei ist es ganz wichtig, daß Sie die y-Koordinate jeweils zuerst angeben, womit der Aufruf beispielsweise folgende Form erhält:

```
Screen+[25,80]
```

Der Aufruf in Zeile 40

```
var Color: Boolean;
```

setzt je nach Art der Grafikkarte die Werte »True« oder »False« und gibt zurück, ob der aktuelle Textmodus farbig oder monochrom ist.

Im Grafikmodus funktioniert diese Abfrage nicht, dort wird immer »True« zurückgegeben. Die Arbeitsweise der Utility führt Ihnen »utildemo.pas« (Listing 2) vor. Utildemo enthält zusätzlich die folgenden nützlichen Zusatzroutinen:

- »Center« gibt eine Zeichenkette zentriert auf dem Bildschirm aus.
- »CopyChr« liefert das Zeichen zurück, das an einer bestimmten Position auf dem Bildschirm steht.

- Normalerweise sollten Sie den Mauscursor vor jeder Ausgabeoperation abschalten und nachher wieder einschalten, weil ein versehentlich »überschriebener« Mauscursor bei einer Mausbewegung den alten Inhalt des Bildschirmspeichers vor der Ausgabeoperation wieder zu »restaurieren« versucht und damit den neuen Text zerstören würde. Der Übersichtlichkeit zuliebe verzichtet Utildemo jedoch hier darauf, weil der Mauscursor, da er sich nur im unteren Rahmen bewegen darf, den Text nicht überschreiben kann.

»SetMouseRange« hält nämlich die Maus in den letzten drei Bildschirmzeilen. Das Zeichen an der Mausposition wird jeweils mit »CopyChr« angezeigt, ebenso die Mausposition, der Status der Maustasten und der Statustasten. Sie beenden das Programm durch simultanes Drücken der linken und der rechten Maustaste, wozu Sie aber auch [Strg-Untbr] einsetzen können.

```
procedure WriteStatus(B: Boolean);
```

(Zeile 22) benutzt das Hauptprogramm, um den Schaltzustand der Statustasten auszugeben.

(Markus Fleck/et)

Bildschirm beschreiben	Bildschirminhalt auslesen
Screen + [y,x].CH: = CH; Screen + [y,x].Att: = Att;	Att: = Screen + [y,x].Att CH: = Screen[y,x] + .CH

Tabelle 2. Mit »var Att: Byte; CH: Char:« manipulieren Sie den Bildschirmspeicher

```
[1X3]    1: {Programm: utility.pas
[4B6]    2:  Funktion: Tpu für Maus, Tastatur etc.
[SZ5]    3:  Sprache : Turbo Pascal ab Version 5.Ø
[193]    4:  Autor: Markus Fleck
[P35]    5:  (c)1991 DMV Widuch GmbH & Co. KG }
[PB2]    6: unit Utility;
[SK1]    7: interface
[KT5]    8: const MaxY=25; MaxX=8Ø; {Bild 8Øx25}
[K24]    9: const HercHi=$ØF; {hell}
[KØ5]   1Ø:       HercHiU=$Ø9; {+Unterstrich}
[PP3]   11:       HercLo=$Ø7; {dunkler}
[CØ5]   12:       HercLoU=$Ø1; {+Unterstrich}
[MG4]   13:       HercInv=$7Ø; {invertiert}
[K25]   14:       Blink=$8Ø; {dazuaddieren für}
[Z86]   15:       {Deklarat. direkt. Bildschirmausgabe}
[5Z6]   16: type ScreenType=array[1..MaxY,1..MaxX] of
[Y11]   17:      record
[213]   18:        CH: Char; ATT: Byte
[ZNØ]   19:      end;
[MØ5]   2Ø: type ScreenPtr=^ScreenType;
[8X3]   21: var Screen: ScreenPtr;
[GV4]   22:     {Status der Sondertasten}
[657]   23: var _Insert,_Shift,_Ctrl,_Alt,_NumLock,
[AH5]   24:     _CapsLock,_ScrollLock: Boolean;
[EG7]   25: procedure ClrKeyBuf; {Tastaturroutinen}
[8F4]   26: procedure GetStatusKeys;
[886]   27: procedure HideCursor; {Cursor ein- }
[FB7]   28: procedure ShowCursor; {und ausschalten}
[VF5]   29: var Mouse: Boolean; {Abfrage +}
[YS7]   3Ø: procedure ResetMouse; {Steuerung der Maus}
[7N3]   31: procedure ShowMouse;
[GH3]   32: procedure HideMouse;
[CK4]   33: function  MouseKeys: Byte;
[E16]   34: procedure GetMouseXY(var x,y: Byte);
[AE5]   35: procedure PutMouseXY(x,y: Byte);
[Ø17]   36: procedure SetMouseRange(MinX,MinY,MaxX,
[1X1]   37:                MaxY: Byte);
[VM3]   38: procedure WaitMouse;
[U45]   39: {direkter Bildschirmzugriff}
[RK6]   4Ø: var Color: Boolean; {Farbe oder Mono?}
[P95]   41: procedure SetBlink(On: Boolean);
[MK4]   42: procedure Border(Col: Byte);
[D65]   43: procedure SetColor(ATT: Byte);
[US6]   44: procedure WriteChar(x,y: Byte; CH: Char);
[4L2]   45: implementation
[S62]   46: uses Dos,Crt;
[UE3]   47: var Reg: Registers;
[E36]   48:     {Tastaturpuffer-Statusvariablen}
[XU5]   49: var Anfang: Byte absolute $ØØ4Ø:$ØØ1A;
[9R4]   5Ø:     Ende: Byte absolute $ØØ4Ø:$ØØ1C;
[MM7]   51: procedure ClrKeyBuf; {Tast.Puffer löschen}
[MUØ]   52: begin
[192]   53:   Anfang:= Ende;
[ZNØ]   54: end;
[HP7]   55: procedure GetStatusKeys; {Statustasten}
[MUØ]   56: begin
[GU1]   57:   Reg.AH:= $Ø2;
[Ø42]   58:   Intr($16,Reg);
[384]   59:   _Insert:= (Reg.AL AND 128)>Ø;
```

Listing 1. »utility.pas« ist die Unit, die Ihnen Kontrolle über Tastatur, Maus- und Cursortasten gibt

```
[KA4]   6Ø:    _Shift:= ((Reg.AL AND 1)>Ø) OR
[VN2]   61:    ((Reg.AL AND 2)>Ø);
[GX3]   62:    _Ctrl:= (Reg.AL AND 4)>Ø;
[3F3]   63:    _Alt:= (Reg.AL AND 8)>Ø;
[MB4]   64:    _NumLock:= (Reg.AL AND 32)>Ø;
[3R4]   65:    _CapsLock:= (Reg.AL AND 64)>Ø;
[RT4]   66:    _ScrollLock:= (Reg.AL AND 16)>Ø;
[ZNØ]   67: end;
[F27]   68: var CursorBeg,CursorEnd: Word; {Crsblck}
[PØ4]   69: procedure StoreCursor;
[MUØ]   7Ø: begin
[AY3]   71:   Reg.AH:= $ØF; Intr($1Ø,Reg);
[RX3]   72:   Reg.AH:= $Ø3; Intr($1Ø,Reg);
[BR6]   73:   CursorBeg:= Reg.CH; CursorEnd:= Reg.cl;
[ZNØ]   74: end;
[TA7]   75: procedure HideCursor; {versteckt Cursor}
[MUØ]   76: begin
[RU3]   77:   Reg.AH:= $Ø1; Reg.CH:= $2Ø;
[9W3]   78:   Reg.CL:= $ØØ; Intr($1Ø,Reg)
[ZNØ]   79: end;
[E58]   8Ø: procedure ShowCursor; {zeigt Cursor wieder}
[MUØ]   81: begin
[CØ5]   82:   Reg.AH:= $Ø1; Reg.CH:= CursorBeg;
[SA5]   83:   Reg.CL:= CursorEnd; Intr($1Ø,Reg)
[ZNØ]   84: end;
[FH7]   85: procedure ResetMouse; {init. Maustreiber}
[MUØ]   86: begin
[DY3]   87:   Reg.AX:= Ø; Intr($33,Reg);
[BG3]   88:   Mouse:= Reg.AX=$FFFF;
[AU3]   89:   if NOT Mouse then Exit;
[MN3]   9Ø:   Reg.AX:= $A; Reg.BX:= $Ø;
[263]   91:   if Color then begin
[EN4]   92:     Reg.CX:= $7FFF; Reg.DX:= $7ØØØ;
[ØC2]   93:   end else begin
[6L4]   94:     Reg.CX:= $ØØØØ; Reg.DX:= $ØFDB;
[ZNØ]   95:   end;
[542]   96:   Intr($33,Reg);
[ZNØ]   97: end;
[WG7]   98: procedure HideMouse; {versteckt die Maus}
[MUØ]   99: begin
[AU3]  1ØØ:   if NOT Mouse then Exit;
[DØ4]  1Ø1:   Reg.AX:= $2; Intr($33,Reg);
[ZNØ]  1Ø2: end;
[3B7]  1Ø3: procedure ShowMouse; {Maus-Block-Cursor}
[WC4]  1Ø4: begin {Aufruf paarweise}
[AU3]  1Ø5:   if NOT Mouse then Exit;
[9Ø4]  1Ø6:   Reg.AX:= $1; Intr($33,Reg);
[ZNØ]  1Ø7: end;
[K78]  1Ø8: function MouseKeys: Byte; {Status Maustast}
[574]  1Ø9: begin {siehe Tabelle 1}
[KU3]  11Ø:   if NOT Mouse then begin
[Z83]  111:     MouseKeys:= Ø; Exit
[ZNØ]  112:   end;
[P24]  113:   Reg.AX:= 3; Intr ($33,Reg);
[R53]  114:   MouseKeys:= Reg.BX
[ZNØ]  115: end;
[E16]  116: procedure GetMouseXY(var x,y: Byte);
[MUØ]  117: begin
[Ø76]  118:   if NOT Mouse then {liefert die Maus-}
[EL6]  119:   begin {koordianten, umgerechnet auf }
[DD5]  12Ø:     x:= 1; y:= 1; {Bildschirmkoordi.}
[ASØ]  121:     Exit
[ZNØ]  122:   end;
[P24]  123:   Reg.AX:= 3; Intr ($33,Reg);
[K63]  124:   x:= (Reg.CX SHR 3)+1;
[133]  125:   y:= (Reg.DX SHR 3)+1
[ZNØ]  126: end;
[3N5]  127: {setzt die Maus an die Stelle (x,y)}
[AE5]  128: procedure PutMouseXY(x,y: Byte);
[EF2]  129: var x2,y2: Word;
[MUØ]  13Ø: begin
[AU3]  131:   if NOT Mouse then Exit;
[PU4]  132:   x2:= (x-1) SHL 3; y2:= (y-1) SHL 3;
[FS3]  133:   Reg.AX:= 4; Reg.CX:= x2;
[Ll4]  134:   Reg.DX:= y2; Intr ($33,Reg)
[ZNØ]  135: end;
[Ø17]  136: procedure SetMouseRange(MinX,MinY,MaxX,
[FW4]  137:                MaxY: Byte); {Maus-Bereich}
[MUØ]  138: begin
[AU3]  139:   if NOT Mouse then Exit;
[GE5]  14Ø:   Reg.AX:= 7; Reg.CX:= (MinX-1) SHL 3;
[FT5]  141:   Reg.DX:= (MaxX-1) SHL 3; Intr ($33,Reg);
[PE5]  142:   Reg.AX:= 8; Reg.CX:= (MinY-1) SHL 3;
[RR5]  143:   Reg.DX:= (MaxY-1) SHL 3; Intr ($33,Reg)
[ZNØ]  144: end;
[RC7]  145: procedure WaitMouse; {alle Maustast. los}
[EP2]  146: var Keys: Word;
[MUØ]  147: begin
[AU3]  148:   if NOT Mouse then Exit;
[P31]  149:   repeat
[P24]  15Ø:     Reg.AX:= 3; Intr ($33,Reg);
[212]  151:     Keys:= Reg.bx
[532]  152:   until Keys=Ø
[ZNØ]  153: end;
[F97]  154: procedure GetScreenType; {Herc-/Farbmod.}
[YT2]  155: var MODE: ^Byte;
[MUØ]  156: begin
[RR2]  157:   MODE:= Ptr($4Ø,$49);
[3P1]  158:   if MODE^= 7
[E54]  159:   then Screen:= Ptr($BØØØ,ØØØØ)
[CC4]  16Ø:   else Screen:= Ptr($B8ØØ,ØØØØ);
[9T4]  161:   Color:= Screen=Ptr($B8ØØ,ØØØØ);
[ZNØ]  162: end;
[K87]  163: procedure SetBlink(On:Boolean); {EGA/VGA!}
[MUØ]  164: begin
[2T3]  165:   if NOT Color then Exit;
[VU3]  166:   Reg.AH:= $1Ø; Reg.AL:= $Ø3;
[WA3]  167:   if On then Reg.BL:= $Ø1
[DW4]  168:   else Reg.BL:= $ØØ; Intr($1Ø,Reg);
[ZNØ]  169: end;
[417]  17Ø: procedure Border(Col: Byte); {Farbrahmen}
[VF3]  171: begin {nur EGA/VGA}
[RG4]  172:   Col:= Col MOD 16; Reg.AH:= $ØB;
[HY5]  173:   Reg.BH:= Ø; Reg.BL:= Col; Intr($1Ø,Reg);
[ZNØ]  174: end;
[DE7]  175: procedure SetColor(ATT: Byte); {Textfarbe}
[7V6]  176: begin {SetColor(VorderGr+HinterGr*16)}
[AY5]  177:   TextAttr:= ATT; {Definiert in Crt}
[ZNØ]  178: end;
[US6]  179: procedure WriteChar(x,y: Byte; CH: Char);
[PØ4]  18Ø: begin { siehe Tabelle 2}
[TK3]  181:   Screen^[y,x].CH:= CH;
[225]  182:   Screen^[y,x].ATT:= TextAttr;
[ZNØ]  183: end;
[G77]  184: procedure WriteLine(x,y,L: Byte; CH: Char);
[MUØ]  185: begin
[H43]  186:   for x:= x to x+L-1 do
[L75]  187:   if x<MaxX then WriteChar(x,y,CH);
[ZNØ]  188: end;
[SZ3]  189: begin {Hauptprogramm}
[N67]  19Ø:   StoreCursor; GetScreenType; ResetMouse;
[KNØ]  191: end.
```

Listing 1. »utility.pas« ist die Unit, die Ihnen Kontrolle über Tastatur, Maus- und Cursortasten gibt

```
[G74]    1: {Programm: utildemo.pas
[G35]    2:  Funktion: Demo zur Tpu-Utility
[SZ5]    3:  Sprache : Turbo Pascal ab Version 5.Ø
[193]    4:  Autor: Markus Fleck
[TX4]    5:  (c)1991 DMV Widuch GmbH & Co. KG}
[U13]    6: program UtilDemo;
[H13]    7: uses Crt,Utility;
[NG5]    8: var MouseX,MouseY,MouseK: Byte;
[M56]    9: procedure Center(S: string; L: Byte);
[PY1]   1Ø: var i: Byte;
[217]   11: begin {String S länger als 8Ø Zeichen?}
[T84]   12:   if S[Ø]>#8Ø then S[Ø]:= #8Ø;
[HV3]   13:   for i:= 1 to Length(S) do
[1G5]   14:   WriteChar((MaxX-Length(S)) DIV 2+i,
[5V3]   15:   L,S[i]); {...ausgeben}
[ZNØ]   16: end;
[ZG5]   17: function CopyChr(X,Y:Byte): Char;
[MUØ]   18: begin
[996]   19:   HideMouse; CopyChr:= Screen^[Y,X].CH;
[UV1]   2Ø:   ShowMouse;
[ZNØ]   21: end;
[BT5]   22: procedure WriteStatus(B:Boolean);
[MUØ]   23: begin
[PW5]   24:   ClrEoL; {Rest der Zeile löschen}
[6HØ]   25:   if B
[ME3]   26:   then WriteLn('+  A N  +')
[PF5]   27:   else WriteLn('- A U S -'); WriteLn;
[ZNØ]   28: end;
[617]   29: var LoopX,LoopY: Byte; {Schlf.Variablen}
[SZ3]   3Ø:     begin {Hauptprogramm}
[NV5]   31:       if Color then SetColor(White+Red*16)
[LH6]   32:       else SetColor(HercInv); {mono-invers}
[1Z5]   33:       ClrScr; {VGA-> grün; Herc.-> grau}
[1Z3]   34:       SetMouseRange(1,23,8Ø,25);
[KT5]   35:       PutMouseXY(4Ø,24); {Maus Pos.: 4Ø,24}
[SX4]   36:       Center('U T I L I T Y  D E M O',2);
[7V3]   37:       Screen^[25,8Ø].CH := 'X';
[6M6]   38:       ClrKeyBuf; {Tastaturpuffer löschen}
[C16]   39:       HideCursor; {Cursor verstecken}
[M32]   4Ø:       if Color then
[5L6]   41:       TextAttr:= White+Blue*16 {weiß blau}
[CG5]   42:       else TextAttr:= HercLo; {grau}
[UL3]   43:       for LoopY:= 23 to 25 do
[2G3]   44:       for LoopX:= 1 to 8Ø do
[CY3]   45:       WriteChar(LoopX,LoopY,
[2Ø3]   46:       Chr(Random(2ØØ)+4Ø));
[696]   47:       ShowMouse; {Mauscursor, falls vorh.}
[P23]   48:       repeat GotoXY(1,5);
[VK2]   49:       GetStatusKeys;
[YA3]   5Ø:     Write('  INSERT:     ');
[YY3]   51:     WriteStatus(_Insert);
[Ø73]   52:     Write('  SHIFT:      ');
[KP3]   53:     WriteStatus(_Shift);
[Z13]   54:     Write('  CTRL:       ');
[LL3]   55:     WriteStatus(_Ctrl);
[9X2]   56:     Write('  ALT:        ');
[LA3]   57:     WriteStatus(_Alt);
[9C3]   58:     Write('  NumLock:    ');
[L34]   59:     WriteStatus(_NumLock);
[SD3]   6Ø:     Write('  CapsLock:   ');
[C94]   61:     WriteStatus(_CapsLock);
```

Listing 2. »utildemo.pas« führt Ihnen die Arbeit der Unit »Utility« vor

Noch erhältlich:

DOS EXTRA Nr. 3

- Datenfernübertragungsprogramm
- Hercules-Grafik-Toolbox in Turbo Pascal
- Grafikfunktionen für Turbo C
- Zeichensatzeditor für Epson-Drucker
- Treiber für AT-Tastatur
- Alle Listings zum Kurs »PC-glasklar«
- Demo für Sprachdigitalisierung

DOS EXTRA Nr. 4

- Desktop-Publishing-Programm »SDTP«
- Kommandoeditor für MS-DOS
- Editor für Disketten und Festplatten
- Speicherresidente Programme verwalten
- Paßwort-Schutz für MS-DOS
- Tastatureingabe automatisiert
- Clock-Utility für alle Grafikkarten
- Cursor und Zeichensatz für EGA-Karte
- Fenstertechnik für Turbo C
- Generator für Hilfsbildschirme

DOS EXTRA Nr. 5

- 16 Farben für die CGA-Farbgrafikkarte
- Snapshot: Bildschirmfotos per Software
- Druckerzeichensatz selbstgemacht
- Mausgesteuertes Zeichenprogramm
- Editor in GW-Basic
- Setup-Programm für ATs
- Maschinensprache für GW-Basic
- Erklärtes Listing: das Basic-Spiel »Stardos«

DOS EXTRA Nr. 6

- SBasic-Compiler im Quelltext
- Datenbankprogramm »Parat«
- Fensterfunktionen für GW-Basic und SBasic
- Turbo-Basic-Bibliothek für 16 Farben
- Analyse von Grafikkarten
- Maskengenerator für Binärdateien
- dBase-Datenbanken in Basic
- Superhirn für Quick Basic 4.0

DOS EXTRA Nr. 7

- MS-DOS-Benutzeroberfläche Easydos
- Benutzeroberfläche für SBasic
- Datengenerator für Basic, C und Pascal
- Sprite-Editor für Farbgrafikkarten
- DOS-Packer
- Virenschutzprogramm
- Weltkarte im Computer
- Einfacher Prozessor-Emulator

DOS EXTRA Nr. 8

- Makroassembler SAM III
- Exe2bin-Ersatz durch "exe2com"
- Speicherverwaltung unter MS-DOS
- Routine zum Speichern von Hardcopies

DOS EXTRA Nr. 10

- Units-Bibliotheken für Turbo Pascal: Datum, Sound, Bildschirmausgabe
- Snapshot-Utility, Version 2.7
- TSR-Datei-Manager
- DOS-Assembler SAM III, Version 1.1
- Schnelles Kopierprogramm
- Komfortable Dateiumleitung
- Hercules-Grafik in Basic programmiert
- Patch für BGI-Treiber von Turbo Pascal

DOS EXTRA Nr. 11

- Preiswerte HPGL-Erweiterung, der HPGL-Extender
- Leistungsstarker Diskетteneditor
- Unit für die komfortable Speicherverwaltung
- Blockbefehle für den Turbo-Pascal-Editor
- Sound-Erzeugung unter Turbo Pascal
- Speicherresidente Uhr

DOS EXTRA Nr. 12

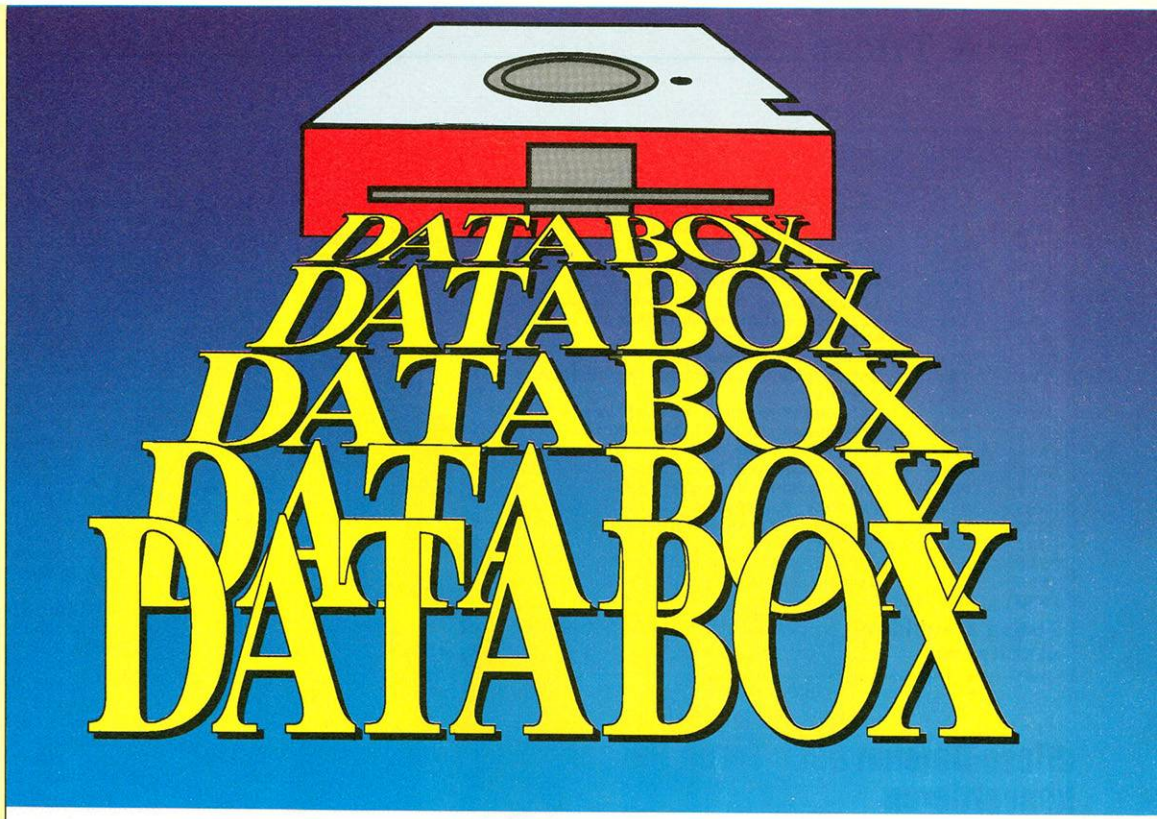
- Kreative Grafik in Basic zum Mitmachen
- Grafik-Anpassung von Basic-Programmen
- Außergewöhnliche Fraktalgrafiken
- Das ultimative Hardcopy-Utility
- Testbild für Monitore in Basic programmiert

DOS EXTRA Nr. 14

- komfortables Datensicherungsprogramm
- Zeichensatzeditor für 24-Nadel-Drucker
- Pascal-Quelltexte optimieren
- Utilities zur Bearbeitung von Dateien und Verzeichnissen

DOS EXTRA Nr. 15

- der Nachbrenner für GW-Basic: SBasic Version 2.0
- der Tastaturtreiber »SKeybgr« in der Version 4.0
- »Schiffe versenken« mit einem Quick-Basic-Programm
- Simulation von Tastatureingaben mit »Prokey«



Die Software zu den DOS-EXTRA-Heften

Sagenhaft günstig, extra viel gute Software fürs Geld!

DATABOX:

- **mehr als der übliche Softwareservice**
- **bringt alle Programme der DOS EXTRA auf Datenträger**

Für alle PCs unter MS-DOS

Den Aufbau der Databox-Diskette finden Sie in der Datei »read.me« erklärt. Geben Sie deshalb vor dem Start »type read.me« ein. Sämtliche Pascal- und C-Programme liegen auch in einer compilierten Form für Hercules- und Farbgrafikkarten vor. Es ist allerdings nicht gewährleistet, daß diese Versionen auf Hardware, die nicht vollständig kompatibel zum IBM PC ist, problemlos laufen. In diesem Fall müssen Sie den Quellcode nach der Anpassung mit dem entsprechenden Compiler neu übersetzen.
Die Basic-Programme arbeiten mit dem GW-Basic-Interpreter von Microsoft einwandfrei. Bei Interpretern und Compilern anderer Hersteller kann es zu Problemen bei der Programmausführung kommen, sofern diese nicht voll kompatibel sind.

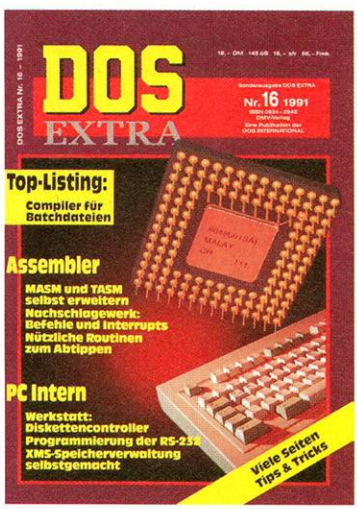
jeweils 2 Disketten 5 1/4“
1 Diskette 3 1/2“

35,– DM (unverbindliche Preisempfehlung)
Wenn Sie über den DMV-Bestellservice bestellen, gilt folgendes:

Inland:		**Ausland:**	
Einzelpreis	35,– DM	Einzelpreis	35,– DM
zzgl. Versandkosten	4,– DM	zzgl. Versandkosten	6,– DM
Endpreis	**39,– DM**	**Endpreis**	**41,– DM**

Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland 4,– DM bzw. für das Ausland 6,– DM Porto und Verpackung.

– Bitte benutzen Sie die Bestellkarte. –



DOS EXTRA Nr. 16

- Batch-Compiler der S-Klasse
- Config-Editor beim Booten
- Assembler-Preprozessor für MASM und TASM
- GEM-API-Unit für Turbo Pascal
- SCheck-Prüfsummenprogramm in der Version 3.0

DMV-Verlag · Postfach 250 · 3440 Eschwege

DMV Daten- und Medienverlag

```
[GP3]    62:       Write('  ScrollLock: ');
[GM4]    63:       WriteStatus(_ScrollLock);
[8P4]    64:       GetMouseXY(MouseX,MouseY);
[A42]    65:       GotoXY(1Ø,19);
[E46]    66:       Write('Mausposition:'); GotoXY(25,19);
[3U4]    67:       Write(MouseX,#32'-'#32,MouseY);
[316]    68:      GotoXY(42,19); Write('Zeichen an der');
[FT4]    69:      Write(#32'Mausposition --> [ ');
[SE6]    7Ø:      {Zeichen an der Mauspos. ausgeben:}
[V75]    71:      Write(CopyChr(MouseX,MouseY));
[M22]    72:      Write(#32']');
[ÉE3]    73:      MouseK := MouseKeys;
[KR5]    74:      GotoXY(1Ø,21); Write('Maustasten: ');
[5B1]    75:      ClrEoL;
[R15]    76:      if MouseK=Ø then Write('keine');
[516]    77:      if MouseK AND 1=1 then Write('links ');
[A16]    78:      if MouseK AND 4=4 then Write('Mitte ');
[R26]    79:      if MouseK AND 2=2 then Write('rechts');
[156]    8Ø:      until (MouseKeys=3); {bis linke und}
[RE6]    81:      {rechte Maustaste gedrückt werden}
[BC6]    82:      ResetMouse; {Mausfenster und andere
[BØ7]    83:      Maus-Statusdaten wieder zurücksetzen}
[5A6]    84:      ShowCursor; {Cursor wieder zeigen}
[956]    85:      HideMouse; {Mauscursor verstecken}
[KNØ]    86: end.
```

Listing 2. »utildemo.pas« führt Ihnen die Arbeit der Unit »Utility« vor

dBase-Dateien nach Turbo Pascal konvertieren

Da Turbo Pascal seit dem Erscheinen von Version 4.0 und vor allem Version 6.0 ein weit verbreitetes Produkt ist, finden Sie »convert.pas« (Listing 3) in und für Turbo Pascal. Gerade Programmierer wollen ja nicht nur dBase-kompatible Dateien bearbeiten, was durch eine Vielzahl von Werkzeugen heute realisierbar ist, sondern darüber hinaus verstehen, wie dBase-Dateien aufgebaut sind. So soll dieses Programm als Anregung dienen, selbst mit fremden Dateiformaten zu experimentieren.

Für die Entwicklung des Programms benutzen Sie Turbo Pascal 6.0. Es wurden jedoch keine Routinen eingesetzt, die nicht auch frühere Versionen unterstützen. Deshalb kompilieren Sie Convert auch mit Version 5.5 anstandslos.

Convert rufen Sie mit einem Parameter auf, mit dem Sie den Namen der zu konvertierenden Datei übergeben, beispielsweise:

```
convert aktien.dbf
```

Geben Sie den Befehl am besten im aktuellen Verzeichnis ein, da ansonsten die Bezeichnung des Datensatzes, wie Sie ihn später im Text erwähnt finden, den Pfad beinhaltet. Nach Programmstart versucht Convert zunächst alle betreffenden Dateien zu öffnen: Quell-, Ziel- und Textdatei. War dies erfolgreich, so liest Convert den Dateikopf (Fileheader) der dBase-Datei. Bei einer Kennziffer im ersten Byte des Headers, die darauf schließen läßt, daß es sich um keine dBase-kompatible Datei handelt, gibt Convert eine entsprechende Meldung aus, bricht jedoch die Arbeit nicht ab. Der Hintergrund dafür ist, daß Sie in der Praxis vielleicht schon Dateien zu bearbeiten hatten, bei denen das erste Byte nicht auf eine dBase-Datei deutete, obwohl die Datei ansonsten allen dBase-Konventionen genügte. Nach einer Prüfung auf ausreichenden Speicherplatz im aktuellen Laufwerk schreibt Convert nun den Dateikopf (Recorddeklaration) für die Turbo-Pascal-Datei in eine gesonderte Textdatei, welche die gleiche Bezeichnung wie die Ursprungsdatei jedoch mit der Endung »rec« (Listing 4) aufweist. Anschließend liest Convert alle Datensätze und schreibt diese in die Turbo-Pascal-Datei – wiederum gleichen Namens, doch diesmal mit der Endung »tpf«. Für das Programm, welches später die Datei bearbeiten soll, können Sie diese Satzbezeichnung einfach über die Funktion [Block lesen] des Turbo-Pascal-Editors in den Quellcode einlesen. Bei der Konvertierung werden allerdings Indexdateien und Memofelder nicht berücksichtigt.

(Rolf Schettler/et)

```
[5X3]     1: {Programm: convert.Pas
[676]     2:  Funktion: konvertiert DBASE-Dateien
[ØS2]     3:            in PASCAL-Dateien
[FM3]     4:  Autor: Rolf Schettler
[KØ6]     5:  Sprache  :Turbo Pascal ab Version 5.Ø
[6V4]     6:  (c)1991 DMV Widuch GmbH & Co.KG}
[ZC5]     7: program Convert ; uses Crt,Dos;
[437]     8: type {Record für allg. Dateiinfo, nimmt
[TR5]     9:   die ersten 32 Byte des Headers auf}
[P62]    1Ø:   Header=record
[XH5]    11:   VersionNo: Byte; {Filekennung}
[X54]    12:   ChgYear: Byte; {Datum       }
[Z25]    13:   ChgMonth: Byte; {der letzten}
[TZ4]    14:   ChgDay: Byte; {Änderung    }
[7X6]    15:   RecCount: Longint;{Anzahl Datensätze}
[RD7]    16:   HeaderSize: Integer;{Größe des Headers}
[T77]    17:   RecLength: Integer;{Länge eines Satzes}
[E77]    18:   Reserved1: Integer;{Reserviert für DBase}
[ØV6]    19:   Command: Byte;{Befehlsdurchführung}
[DE6]    2Ø:   Crypt: Byte;{File verschlüsselt ?}
[ØZ4]    21:   Network: array [1..12] of Byte;
[B36]    22:   MDX: Byte;{zugehörige MDX-Datei?}
[G77]    23:   Reserved2: Integer;{Reserviert für DBase}
[DD7]    24:   Reserved3: Byte; {Reserviert für DBase}
[ZNØ]    25: end;
[EH6]    26:       {Record für die Informationen jedes
[1Ø6]    27:       einzelnen Feldes eines Satzes, nimmt
[WC3]    28:       jeweils 32 Byte auf}
[RD3]    29:       FieldPoint= ^Field;
[523]    3Ø:       Field=record Name:
[ZG3]    31:       array [1..1Ø] of Byte;
[RB6]    32:       {Feldname, 1Ø Byte gefüllt mit Ø Byte}
[9P5]    33:         Delimiter: Byte;{Ø Byte Endmarke}
[9E5]    34:         FType: Char;{Feldtyp D,C,F,L,M,N}
[6U5]    35:         Reserved1: Longint;{Rerserviert}
[CS4]    36:         FLength: Byte;{Feldlänge}
[ZL5]    37:         Places: Byte;{Nachkommastelle}
[YU5]    38:         Reserved2: Integer;{Reserviert}
[7R5]    39:         Selected: Byte;{Arbeitsbereich}
[W85]    4Ø:         Reserved3: array [1..1Ø] of Byte;
[YS4]    41:         Key: Byte;{Key angelegt ?}
[XW2]    42:         Next: FieldPoint;
[Y96]    43:         {Next gehört nicht mehr zum Header,
[B96]    44:         dient zum Aufbau verketteter Liste}
[ZNØ]    45:       end;
[KP3]    46: var FName: string[67];
[UT4]    47:     N: Word; FieldCount: Integer;
[AW5]    48:     ReadRecs: Integer; SourceFile: file;
[5Ø5]    49:     DestFile: file; RecFile: Text;
[SY2]    5Ø:     DateiKopf: Header;
[HY3]    51:     AZ,HZ,Feld: FieldPoint;
[P73]    52:     HeapBegin: pointer;
[466]    53:     {Nachfolgende Funktion bildet aus den
[GC6]    54:     Einträgen des Datensatzes Strings im
[6K2]    55:     Pascal Format}
[GF4]    56: function ConvertToString
[TH6]    57: (FieldName: pointer;Leng:Byte):string;
[S56]    58: var Offset: Integer; Namen: string;
[MUØ]    59: begin
[M43]    6Ø:   Offset:=Ø; Namen:='';
[ØU3]    61:   while (Offset<Leng) AND
[276]    62:   (Mem[Seg(FieldName^): Ofs(FieldName^)
[UC3]    63:   +Offset]<>Ø) do begin
[8A6]    64:     Namen:=Namen+Chr(Mem[Seg(FieldName^):
[NA4]    65:     Ofs(FieldName^)+Offset]);
[GZ1]    66:     Inc(Offset);
[ZNØ]    67:   end;
[5A4]    68:   ConvertToString:=Namen;
[ZNØ]    69: end;
[B66]    7Ø: {Die nächste Prozedur schreibt den
[V66]    71: zur konvertierten Datei gehörenden
[U44]    72: Record in eine Textdatei}
[RX3]    73: procedure WriteRecord;
[ØZ1]    74: var N: Byte;
[PC6]    75:     VarName: string; VarType: string[32];
[AS3]    76:     NumString:string[3];
[MUØ]    77: begin
[Y56]    78:   New(AZ);           {Alle Felder werden in}
[S37]    79:   AZ^.Next:=NIL;     {eine gekettete Liste }
[DØ5]    8Ø:   Feld:=AZ;          {eingetragen}
[UA7]    81:  {Lesen der allgemeinen Dateiinformationen}
[ED5]    82:   BlockRead(SourceFile,DateiKopf,
[BZ1]    83:             32,ReadRecs);
[ZC2]    84:   if DiskFree(Ø)<
[N37]    85:   FileSize(SourceFile)-DateiKopf.HeaderSize
[ZN1]    86:   then begin
[TF5]    87:     WriteLn('Zu wenig freier Platz'+
[K54]    88:     'im aktuellen Laufwerk');
[1W5]    89:     Close(SourceFile); Erase(DestFile);
[DUØ]    9Ø:     Halt;
[ZNØ]    91:   end;
[5U5]    92:   WriteLn('Datei wird konvertiert,'+
[NC4]    93:   ' Abbruch mit bel. Taste...');
[187]    94:   {Prüfen auf DBase Kompatibilität anhand
[HH4]    95:   des ersten Kennungsbytes}
[C76]    96:   if  DateiKopf.VersionNo IN [$Ø3,$83,$8B]
[YF1]    97:   then else
[NB6]    98:   Write('Vermutlich Keine DBase-Datei !');
[XX4]    99:   FieldCount:=Round((DateiKopf.
[1Y2]   1ØØ:               HeaderSize-33)/32);
[675]   1Ø1:   for N:= 1 to FieldCount do begin
```

Ein starkes Heft

Im ABO

Es lohnt sich!

Bitte benutzen Sie die Bestellkarte

DMV-Verlag • Postfach 250 • 3440 Eschwege

DMV Daten- und Medienverlag

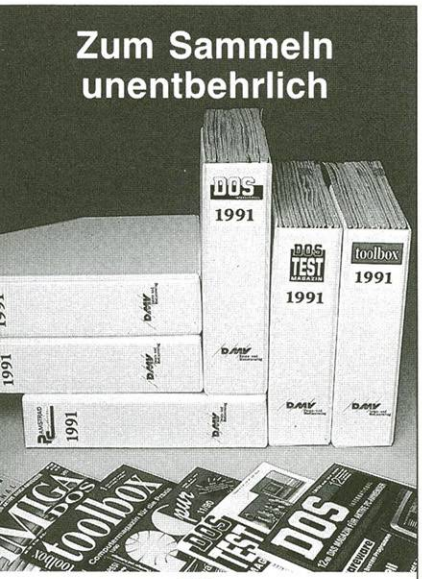
```
[W47]  1Ø2:     {Lesen der einzelnen Feldinformationen}
[DR6]  1Ø3:     BlockRead(SourceFile,Feld^,32,ReadRecs);
[3S6]  1Ø4:     New(HZ); HZ^.Next:=NIL; Feld^.Next:=HZ;
[AF1]  1Ø5:     Feld:=HZ;
[ZNØ]  1Ø6:   end;
[U36]  1Ø7:   {Schreiben des Records in Textdatei}
[EX4]  1Ø8:   WriteLn(RecFile,Copy(FName,1,
[ZF4]  1Ø9:   Pos('.',FName)-1),' = Record');
[6F5]  11Ø:   {erstes Byte ist Löschmarke}
[GC6]  111:   WriteLn(RecFile,'     DELETED:Boolean;');
[2A6]  112:   HZ:= AZ; while HZ^.Next <> NIL do begin
[VY2]  113:     with HZ^  do begin
[174]  114:       Str(FLength,NumString);
[Ø36]  115:       {Folg. Routine ermittelt anhand des
[666]  116:       Feldtyps und der Länge die passende
[HØ3]  117:       Pascalvariable}
[Z22]  118:       case FType of
[2A6]  119:      'C':VarType:='String['+NumString+'];';
[N16]  12Ø:      'N':if Places<> Ø then VarType:='Real;'
[KT3]  121:          else if FLength<=6then
[RL4]  122:          VarType:='Integer;' else if
[NY5]  123:          FLength<=11then VarType:='Longint;'
[ZM3]  124:          else VarType:='Real;';
[PC3]  125:          'F': VarType:='Real;';
[PY3]  126:          'L': VarType:='Boolean;';
[AE4]  127:          'M': VarType:='String[1Ø];'+
[MU3]  128:          '  {Memo,nicht lesbar}';
[ZD4]  129:          'D': VarType:='String[1Ø];';
[ZNØ]  13Ø:       end;
[D36]  131:       VarName:=ConvertToString(@Name,1Ø);
[U66]  132:       {Prüfung auf unerlaubte Zeichen im
[5S2]  133:       Variablennamen }
[RY4]  134:       for N:=1 to Length(VarName) do
[UP2]  135:       if VarName[N] IN
[W57]  136:       ['ä','Ä','ö','Ö','ü','Ü','ß'] then
[RR2]  137:       VarName[N]:='_';
[194]  138:       WriteLn(RecFile,VarName:
[FK2]  139:                              12,' : ',VarType);
[WC2]  14Ø:       HZ:=HZ^.Next;
[ZØ3]  141:     end; {With HZ^}
[VØ4]  142:   end; {While HZ^.Next}
[8Y3]  143:   WriteLn(RecFile,'End;');
[ZNØ]  144: end;
[ØE6]  145: {Nachfolgende Routine konvertiert die
[K75]  146: DBasedatei in eine Pascaldatei}
[ZV3]  147: procedure ConvertFile;
[YL2]  148: var RealV: Real;
[VB6]  149:     IntegerV: Integer; LongintV: Longint;
[GV4]  15Ø:     BooleanV: Byte; Err: Integer;
[VS3]  151:     DatString: string[1Ø];
[YX5]  152:     NumString: string; Buffer: pointer;
[ZL5]  153:     N: Longint; BufferPoint: Integer;
[MUØ]  154: begin
[7M6]  155:   {Puffer für Datensatz reservieren}
[ZZ5]  156:   GetMem(Buffer,DateiKopf.RecLength);
[PØ6]  157:   {Trennmarke zwischen Dateikopf und
[G15]  158:   Datensätzen überspringen}
[PR6]  159:   Seek(SourceFile,FilePos(SourceFile)+1);
[KM6]  16Ø:   for N:=1 to DateiKopf.RecCount do begin
[6B6]  161:     {Einlesen des Datensatzes in Puffer}
[125]  162:     BlockRead(SourceFile,Buffer^,
[M55]  163:     DateiKopf.RecLength,ReadRecs);
[3A5]  164:     {Konvertieren der Löschmarke}
[G46]  165:     if Mem[Seg(Buffer^):Ofs(Buffer^)]=$2Ø
[1P6]  166:     then Mem[Seg(Buffer^):Ofs(Buffer^)]:=Ø;
[GS4]  167:     {Schreiben der Löschmarke}
[BG5]  168:     BlockWrite(DestFile,Buffer^,1);
[FR2]  169:     BufferPoint:=1;
[Z91]  17Ø:     HZ:=AZ;
[TT3]  171:     while HZ^.Next <> NIL do
[MUØ]  172:     begin
[T14]  173:       if KeyPressed then Exit;
[VY2]  174:       with HZ^  do begin
[6V5]  175:         {Konvertieren ASCII->in Pascal}
[232]  176:         NumString:=
[5Z5]  177:         ConvertToString(Ptr(Seg(Buffer^)
[KX5]  178:         ,Ofs(Buffer^)+BufferPoint),FLength);
[LS5]  179:         {Folgende Routine konvertiert das
[DW5]  18Ø:         Stringformat ins Variablenformat}
[DH3]  181:         case FType of 'C': begin
[1C5]  182:             BlockWrite(DestFile,NumString,
[GV1]  183:             FLength+1);
[ZNØ]  184:           end;
[Z54]  185:           'N':if Places<> Ø then begin
[9F4]  186:             Val(NumString,RealV,Err);
[GH4]  187:             BlockWrite(DestFile,RealV,
[WG2]  188:             SizeOf(RealV));
[4MØ]  189:           end
[MY4]  19Ø:           else if FLength <=6 then begin
[9V4]  191:             Val(NumString,IntegerV,Err);
[155]  192:             BlockWrite(DestFile,IntegerV,
[Y33]  193:             SizeOf(IntegerV));
[4MØ]  194:           end
[X45]  195:           else if FLength <=11 then begin
[5W4]  196:             Val(NumString,LongintV,Err);
[V55]  197:             BlockWrite(DestFile,LongintV,
[L43]  198:             SizeOf(LongintV));
[ØC2]  199:           end else begin
[9F4]  2ØØ:             Val(NumString,RealV,Err);
[GH4]  2Ø1:             BlockWrite(DestFile,RealV,
[WG2]  2Ø2:             SizeOf(RealV));
[ZNØ]  2Ø3:           end;
```

```
[LE1]  2Ø4:           'F': begin
[9F4]  2Ø5:           Val(NumString,RealV,Err);
[GH4]  2Ø6:             BlockWrite(DestFile,RealV,
[WG2]  2Ø7:             SizeOf(RealV));
[ZNØ]  2Ø8:           end;
[5F1]  2Ø9:           'L': begin
[4G3]  21Ø:             if NumString='T' then
[ØT4]  211:             BooleanV:=1 else BooleanV:=Ø;
[935]  212:             BlockWrite(DestFile,BooleanV,
[G33]  213:             SizeOf(BooleanV));
[ZNØ]  214:           end;
[8F1]  215:           'M': begin
[1C5]  216:             BlockWrite(DestFile,NumString,
[GV1]  217:             FLength+1);
[ZNØ]  218:           end;
[DE1]  219:           'D': begin
[E12]  22Ø:             DatString:=
[EP3]  221:             Copy(NumString,7,2)+'.'+
[CP3]  222:             Copy(NumString,5,2)+'.'+
[Z93]  223:             Copy(NumString,1,4);
[XH3]  224:             BlockWrite(DestFile,
[882]  225:                        DatString,11);
[4MØ]  226:           end
[ZNØ]  227:         end;
[LD4]  228:         Inc(BufferPoint,FLength);
[ZNØ]  229:       end;
[WC2]  23Ø:       HZ:=HZ^.Next;
[5M3]  231:     end;{While HZ^.Next}
[Ø93]  232:   end; {For N:=1 to }
[ZNØ]  233: end;
[MUØ]  234: begin
[DB1]  235:   ClrScr;
[RF6]  236:   if ParamCount>Ø then FName:=ParamStr(1)
[WP1]  237:   else begin
[WZ4]  238:     WriteLn('Aufruf wie folgt : '+
[PF3]  239:     ' CONVERT DATEINAME');
[DUØ]  24Ø:     Halt;
[ZNØ]  241:   end;
[Z74]  242:   Assign(SourceFile,FName);
[7L4]  243:   Assign(RecFile,Copy(FName,1,
[CC3]  244:   Pos('.',FName))+'Rec');
[4W4]  245:   Assign(DestFile,Copy(FName,1,
[1E3]  246:   Pos('.',FName))+'TPF');
[1E4]  247:   {$I-} Reset(SourceFile,1);
[R74]  248:   if IOResult <>Ø then begin
[T43]  249:     Write('Quelldatei '+
[9E5]  25Ø:     'kann nicht geöffnet werden..');
[DUØ]  251:     Halt;
[ZNØ]  252:   end;
[8C3]  253:   Rewrite(DestFile,1);
[U83]  254:   if IOResult <>Ø then
[MUØ]  255:   begin
[513]  256:     Write('Zieldatei '+
[9E5]  257:     'kann nicht geöffnet werden..');
[633]  258:     Close(SourceFile);
[DUØ]  259:     Halt;
[ZNØ]  26Ø:   end;
[TY2]  261:   Rewrite(RecFile);
[R74]  262:   if IOResult <>Ø then begin
[G33]  263:     Write('Textdatei '+
[9E5]  264:     'kann nicht geöffnet werden..');
[1W5]  265:     Close(SourceFile); Erase(DestFile);
[DUØ]  266:     Halt;
[PS1]  267:   end; {$I+}
[SY4]  268:   Mark(HeapBegin); WriteRecord;
[9F5]  269:   ConvertFile; Release(HeapBegin);
[XW5]  27Ø:   Close(SourceFile); Close(DestFile);
[1K2]  271:   Close(RecFile);
[256]  272:   WriteLn('-Datei ist konvertiert-');
[KNØ]  273: end.
```

Listing 3. »convert.pas« organisiert Ihre dBase-Dateien für Turbo Pascal um

```
[YK2]    1: aktien = Record
[2Y2]    2:      DELETED : Boolean;
[DT2]    3:    KAUFPREIS : Real;
[YT2]    4:       STUECK : Integer;
[BX2]    5:         NAME : String[2Ø];
[944]    6:    KAUFDATUM : String[1Ø];
[A72]    7:       JANUAR : Real;
[XH2]    8:      FEBRUAR : Real;
[HY1]    9:         M_RZ : Real;
[W12]   1Ø:        APRIL : Real;
[MU1]   11:          MAI : Real;
[CW1]   12:         JUNI : Real;
[4W1]   13:         JULI : Real;
[292]   14:       AUGUST : Real;
[CS2]   15:    SEPTEMBER : Real;
[CM2]   16:      OKTOBER : Real;
[4L2]   17:     NOVEMBER : Real;
[2K2]   18:     DEZEMBER : Real;
[2N2]   19:      TENDENZ : Real;
[9U1]   2Ø:         HOCH : Real;
[NU1]   21:         TIEF : Real;
[ZNØ]   22: End;
```

Listing 4. »aktien.rec« speichert Convert auf Ihrem Datenträger



Die DMV-Sammelordner

Sonderhefte • Sammelordner • Demonstrationen

Ausgabe DOS EXTRA DM 18,–	DATABOX 2 Stk. 5 1/4"-Disk DM 35,–	DATABOX 1 Stk. 3 1/2"-Disk DM 35,–
☐ 356 Nr. 3	☐ 357	☐ 358
☐ 381 Nr. 4	☐ 3811	☐ 3812
☐ 382 Nr. 5	☐ 3813	☐ 3814
☐ 383 Nr. 6	☐ 3815	☐ 3816
☐ 384 Nr. 7	☐ 3817	☐ 3818
☐ 385 Nr. 8	☐ 3819	☐ 3820
☐ 387 Nr. 10	☐ 3823	☐ 3824
☐ 388 Nr. 11	☐ 3825	☐ 3826
☐ 389 Nr. 12	☐ 3827	☐ 3828
☐ 391 Nr. 13	———	———
☐ 392 Nr. 14	☐ 3829	☐ 3830
☐ 393 Nr. 15	☐ 3831	☐ 3832
☐ 394 Nr. 16	☐ 3833	☐ 3834
WinDOS DM 18,–	**1 Stk. 5 1/4"-Disk** DM 24,–	**1 Stk. 3 1/2"-Disk** DM 24,–
☐ 30200 Nr. 1	Diskette im Heft	———
☐ 30201 Nr. 2	Diskette im Heft	———

DOS-Taschenbücher	5 1/4"-Diskette DM 24,–	3 1/2"-Diskette DM 24,–
BAND 1	☐ 60501	☐ 60502
BAND 3	☐ 60521	☐ 60522
BAND 4	☐ 60531	☐ 60532

Best.-Nr.		DM
☐ 60500	DOS-Taschenbuch Band 1	29,80
☐ 60510	DOS-Taschenbuch Band 2	29,80
☐ 60520	DOS-Taschenbuch Band 3	29,80
☐ 60530	DOS-Taschenbuch Band 4	29,80
☐ 40101	Das große DOS-Profi-Arbeitsbuch	78,–
☐ 40102	Datenfernverarbeitung - professionell	48,–
☐ 40103	Clipper 5.0	58,–
☐ 402	Die Basic2-Toolbox	49,–
☐ 403	Die Basic2-Toolbox, 5 1/4"-Diskette zum Buch	29,–
☐ 482	QuickStart ConText PRO	19,80

Sammelordner 2 Stk. 15,80

☐ 530 DOS
☐ 534 toolbox
☐ 536 DOS TEST
☐ 531 PC International
☐ 535 PCpur
☐ 537 AMIGA DOS

Zwischensumme + Porto und Verpackung
(Inland 4,– DM, Ausland 6,– DM) _____

Demodisketten jeweils DM 5,–

☐ 001 Fraktal 3D + MaskEdit
☐ 005 ConText PRO
☐ 012 GraphBas PC
☐ 003 ERGO 3.0
☐ 007 3D-Draw
☐ 009 Navigator

Gesamtbetrag: _____ DM

PC-Software

vom

DMV Versandservice

- leistungsstark
- praxisbewährt
- preisbewußt

PC-Software

5 1/4"	3 1/2"		DM
2726 ☐	2727 ☐	**ConText 4.0,**	249,–
2419 ☐	2420 ☐	**ConText-Zeichenediltor**	99,–
256 ☐	257 ☐	**PC-VirusDoktor**	99,–
2903 ☐	2904 ☐	**PC-VirusFinder**	149,–
2905 ☐	2906 ☐	**GraphBas PC,** Turbo Basic	99,–
2907 ☐	2908 ☐	**GraphBas PC,** Microsoft Quick Basic	99,–
2522 ☐	2521 ☐	**ERGO 3.0,** DOS-Benutzeroberfläche	199,–
2702 ☐	2703 ☐	**DMV-Show-Manager**	199,–
238 ☐	239 ☐	**MaskEdit Plus** mit einem Treiber	99,–
2613 ☐	2614 ☐	**MaskEdit Plus,** Paket mit zwei Treibern	149,–
2621 ☐	2622 ☐	**MaskEdit Plus,** Paket mit drei Treibern	199,–

Turbo Pascal 3/4/5	Turbo C Microsoft C	Top-Speed Modula-2	Turbo Basic	Quick Basic
☐ 2632	☐ 2633	☐ 2634	☐ 2635	☐ 2636

2720 ☐	2721 ☐	**WinBasic**	299,–
2730 ☐	2731 ☐	**DMV-WindowsTools** SystemInfo	99,–
2732 ☐	2733 ☐	**DMV-WindowsTools** MultiCopy	99,–
2734 ☐	2735 ☐	**DMV-WindowsTools** HexEdit	99,–
2736 ☐	2737 ☐	**DMV-WindowsTools** TextSearch	99,–
2724 ☐	2725 ☐	**ColoRIX**	299,–
6031 ☐	6032 ☐	**ProfEd**	99,–
227 ☐	233 ☐	**BCI DisAsm/I86,** (MS-DOS)	199,–
2738 ☐	2739 ☐	**NeoGraph**	99,–
6007 ☐	6008 ☐	**3D-Draw, Version 1.0**	99,–
6017 ☐	6018 ☐	**3D-Draw Professional**	199,–
6019 ☐	6020 ☐	**Navigator 4.5**	69,–
6025 ☐	6026 ☐	**Lightning 3D, 1.5**	199,–

5 1/4"	3 1/2"		DM
6035 ☐	6036 ☐	**Fontline**	299,–
6027 ☐	6028 ☐	**DOS-CAD 3.0**	199,–
2700 ☐	2701 ☐	**DMV-Faktura**	398,–
2710 ☐	2711 ☐	**DMV-Atelier**	249,–
2712 ☐	2713 ☐	**DMV-Atelier** Zusatz-Fonts	99,–
2714 ☐	2715 ☐	**DMV-Atelier** Zusatz-Symbole	99,–
20600 ☐	20601 ☐	**MultiCopy**	49,–
20602 ☐	20603 ☐	**DataBase**	49,–
20604 ☐	20605 ☐	**RAM Patrouille**	49,–
20606 ☐	20607 ☐	**Tippsi**	49,–
___ ☐	___ ☐	**toolbox Spezial Nr.** _____	35,–
		Kurzbez.: _____	
2361 ☐	2362 ☐	**MAUSALL Plus**	35,–
1371 ☐	1372 ☐	**PC-Spiele-Sampler**	49,–
248 ☐	249 ☐	**Fraktal-Generator 3D,** MS-DOS	69,–
1374 ☐	—	**Populous PC**	69,95
1379 ☐	1380 ☐	**Indianapolis 500**	49,95
1381 ☐	1382 ☐	**Oil Imperium**	39,95
1383 ☐	1384 ☐	**Loom**	69,95
1385 ☐	1386 ☐	**Zak McKracken**	69,95
1387 ☐	1388 ☐	**Maniac Mansion**	69,95
1295 ☐	1296 ☐	**Memo**	69,–
1297 ☐	1298 ☐	**Trademaster**	49,–
1408 ☐	1409 ☐	**Wing Commander**	89,–
1410 ☐	1411 ☐	**The Secret Missions**	39,–
1406 ☐	1407 ☐	**Wing Commander + The Secret Missions**	119,–
1402 ☐	1403 ☐	**The Secret of Monkey Island**	89,–
1404 ☐	1405 ☐	**Wonderland**	99,–

+ Porto/Verpackung
(Inland 4,– DM, Ausland 6,– DM) _____ DM = Gesamtbetrag: _____ DM

ConText 4.0

Update-Service-Karte

Ja, ich möchte das **ConText-4.0-Update** bestellen.

ConText-4.0-Update DM 99,–
zuzügl. Porto + Verpackung (Inland 4,– DM bzw. Ausland 6,– DM) DM _____
Gesamtpreis DM _____

Gewünschtes Diskettenformat:
☐ 5 1/4", Bestellnummer 2728
☐ 3 1/2", Bestellnummer 2729

Als Benutzernachweis füge ich meiner Bestellung bei: die **Original-Titelseite meines ConText-Handbuches** oder meine **Original-Disketten von**
☐ **ConText Pro** oder
☐ **ConText HIGHSCREEN HIGHPAQ** oder
☐ **ConText DMV-POWER-PACK**

Absender:

Name _____

Vorname _____

Straße/Postfach _____

PLZ/Ort _____

Land _____

Mein Zahlungswunsch:
☐ mit beigefügtem Verrechnungsscheck
☐ Lieferung per Nachnahme zuzügl. Nachnahmegebühr (nur innerhalb der BRD)

Datum Unterschrift (bei Minderjährigen der Erziehungsberechtigte)

PC-Sonderhefte

Ja, ich mache von Ihrem Angebot Gebrauch und bestelle die umseitig ausgewählten Produkte. Bitte liefern Sie an folgende Anschrift:

Name

Straße/Postfach

PLZ/ORT

☐ Den Betrag bezahle ich mit dem beigefügten Verrechnungsscheck.

☐ Ich bitte um Lieferung per Nachnahme zuzüglich der Nachnahmegebühr (nur innerhalb der BRD).

Datum Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Bitte ausreichend frankieren

DMV-Verlag
DOS International
Postfach 250

3440 Eschwege

1. Informieren

DOS-Taschenbücher und DOS-EXTRA-Sonderhefte sind eine zuverlässige Informationsquelle für jeden PC-Besitzer. Nutzen auch Sie dieses Wissen, und vertiefen Sie es mit den dazu angebotenen Programmen aus dem Disketten-Service. Den PC beherrschen mit

- **DOS-EXTRA-Sonderheften**
- **DOS-Taschenbüchern**
- **DOS-Disketten-Service**

Zur Erinnerung:
Meine DOS-Sonderhefte, DOS-Taschenbücher, DOS-Disketten wurden bestellt am

_____

PC-Software

Ja, ich mache von Ihrem Angebot Gebrauch und bestelle die umseitig ausgewählten Produkte. Bitte liefern Sie an folgende Anschrift:

Name

Straße/Postfach

PLZ/ORT

☐ Den Betrag bezahle ich mit dem beigefügten Verrechnungsscheck.

☐ Ich bitte um Lieferung per Nachnahme zuzüglich der Nachnahmegebühr (nur innerhalb der BRD).

Datum Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Bitte ausreichend frankieren

DMV-Verlag
DOS International
Postfach 250

3440 Eschwege

2. Aufrüsten

PC-Software vom DMV-Verlag ist unter Kennern bereits ein Begriff und auch ein gutes Angebot für Sie, Ihren PC mit starken und anwendergerechten Applikationen auszustatten. Von der Textverarbeitung bis zum Programmiersystem, beim DMV-Versandservice kaufen Sie günstig ein. Den PC konsequent nutzen mit PC-Software aus dem DMV-Versandservice

- **PC-Anwendungen**
- **PC-Applikationen**
- **PC-Entwicklungssysteme**

Zur Erinnerung:
Meine PC-Software wurde bestellt am

_____

ConText 4.0 ist da!

Update-Service-Karte

Ja, ich möchte das ConText-4.0-Update!

DOS Extra 16'91

DMV-Verlag
Postfach 250

3440 Eschwege

CONTEXT 4.0

Textverarbeitung mit Verstand

_____

Das Update mit Leistungssprung

DOS INTERNATIONAL

Kompetenz

für

PC-Personalcomputer

im

Abonnement

deutlich

günstiger

Abo-Order DOS International

Hiermit bestelle ich **»DOS International«** für mindestens

☐ **12 Ausgaben** ☐ **6 Ausgaben**

Preise: **BRD**	12 Ausgaben	82,50 DM
	6 Ausgaben	42,– DM
Auslandspreise: **Europa**	12 Ausgaben	110,– DM
	6 Ausgaben	55,– DM
Außereuropäisches Ausland	12 Ausgaben	150,– DM
	6 Ausgaben	75,– DM

Lieferung erfolgt ab der nächsterreichbaren Ausgabe.

Abo-Order DOS DATABOX PLUS

Hiermit bestelle ich **»DOS DATABOX PLUS«** für mindestens

☐ **12 Ausgaben** ☐ **6 Ausgaben**

Diskettenformat: ☐ IBM 5 1/4" ☐ IBM 3 1/2"

Inlandspreise: 12 Ausgaben 300,– DM, 6 Ausgaben 150,– DM
Europa: 12 Ausgaben 320,– DM, 6 Ausgaben 160,– DM
Außereurop. Ausland: 12 Ausg. 360,– DM, 6 Ausg. 180,– DM

Lieferung erfolgt ab der nächsterreichbaren Ausgabe.

Lieferanschrift

☐ Geschäft ☐ Privat

Vorname, Name

Firma (nur wenn Lieferadresse)

Straße, Nr./Postfach

PLZ/Ort *(Bitte genaue Anschrift angeben!)*

Dieses Abonnement verlängert sich automatisch um 6 bzw. 12 Ausgaben, wenn es nicht sechs Wochen vor Ablauf schriftlich gekündigt wird.

Datum 1. Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

☐ **Bequem und bargeldlos durch Bankabbuchung**

Bankleitzahl

Konto-Nr./-Inhaber

Geldinstitut

☐ **Gegen Rechnung** – zahlbar innerhalb zwei Wochen nach Erhalt. (Bitte keine Vorauszahlung leisten — Rechnung abwarten.)

Bitte unbedingt auf der Vorderseite eine zweite Unterschrift leisten!

DOS INTERNATIONAL

Information

über

PC-Personalcomputer

sammeln Sie mit

- **DOS-Einzelheften**
- **DATABOX PLUS** mit mehr als 700 KByte Software und Bonusprogramm

DOS-Einzelhefte und DATABOX

Einzelheft Ausgabe 6,50 DM/Stck	DATABOX 5 1/4"-Diskette 24,– DM/Stck	DATABOX 3 1/2"-Diskette 24,– DM/Stck
☐ 3/88	☐	☐
☐ 4/88	☐	☐
☐ 5/88	☐	☐
☐ 6/88	☐	☐
☐ 7/88	☐	☐
☐ 8/88	☐	☐
☐ 9/88	☐	☐
Einzelheft 7,50 DM/Stck		
☐ 10/88	☐	☐
☐ 11/88	☐	☐
☐ 12/88	☐	☐
☐ 1/89	☐	☐
☐ 2/89	☐	☐
☐ 3/89	☐	☐
☐ 4/89	☐	☐
☐ 5/89	☐	☐
☐ 6/89	☐	☐
☐ 7/89	☐	☐
☐ 8/89	☐	☐
☐ 9/89	☐	☐
☐ 10/89	☐	☐
☐ 11/89	☐	☐
☐ 12/89	☐	☐

Einzelheft Ausgabe 7,50 DM/Stck	DATABOX PLUS DATABOX + Bonus 5 1/4" 2 Disketten 24,– DM/Stck	DATABOX PLUS DATABOX + Bonus 3 1/2"-Diskette 24,– DM/Stck
– 1/90	☐	☐
☐ 2/90	☐	☐
☐ 3/90	☐	☐
☐ 4/90	☐	☐
– 5/90	☐	☐
☐ 6/90	☐	☐
☐ 7/90	☐	☐
☐ 8/90	☐	☐
☐ 10/90	☐	☐
☐ 11/90	☐	☐
☐ 12/90	☐	☐
☐ 1/91	☐	☐
☐ 2/91	☐	☐
☐ 3/91	☐	☐
☐ 4/91	☐	☐
☐ 5/91	☐	☐
☐ 6/91	☐	☐
☐ 7/91	☐	☐
☐ 8/91	☐	☐
☐ 9/91	☐	☐

☐ Gratis-Sammelordner bei Bestellung von 6 Einzelheften

+ Porto/Verpackung
(Inland 4,– DM, Ausland 6,– DM) _____ DM Gesamtbetrag _____ DM

DOS INTERNATIONAL

Vorsprung

für

PC-Personalcomputer

mit

DOS EXTRA

im

Abonnement

Abo-Order DOS EXTRA

Hiermit bestelle ich **»DOS EXTRA«** für mindestens

☐ **4 Ausgaben** (vierteljährliche Erscheinungsweise)

Preise: **BRD**	4 Ausgaben 60,– DM
Auslandspreise: **Europa**	4 Ausgaben 70,– DM
Außereuropäisches Ausland	4 Ausgaben 80,– DM

Lieferung erfolgt ab der nächsterreichbaren Ausgabe.

Dauerbezug DOS EXTRA DATABOX

☐ Hiermit bestelle ich **»DOS EXTRA DATABOX«** im Dauerauftrag mit Kündigungsmöglichkeit zum vierteljährlichen Erscheinungstermin.

Achtung: Nur im Bankabbuchungsverfahren möglich!

Preise: **BRD:**	je 39,– DM
Auslandspreise: **Europa:**	je 41,– DM
Außereurop. Ausland:	je 42,– DM

Lieferung erfolgt ab der nächsterreichbaren Ausgabe.

Diskettenformat: ☐ IBM 5 1/4" ☐ IBM 3 1/2"

Lieferanschrift

☐ Geschäft ☐ Privat

Vorname, Name

Firma (nur wenn Lieferadresse)

Straße, Nr./Postfach

PLZ/Ort *(Bitte genaue Anschrift angeben!)*

Dieses Abonnement verlängert sich automatisch um 4 Ausgaben, wenn es nicht sechs Wochen vor Ablauf schriftlich gekündigt wird.

Datum 1. Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

☐ **Bequem und bargeldlos durch Bankabbuchung**

Bankleitzahl

Konto-Nr./-Inhaber

Geldinstitut

☐ **Gegen Rechnung** – zahlbar innerhalb zwei Wochen nach Erhalt. (Bitte keine Vorauszahlung leisten — Rechnung abwarten.)

Bitte unbedingt auf der Vorderseite

DOS INTERNATIONAL

Abonnement

Rechnungsanschrift

Vorname, Name

Straße, Nr./Postfach

PLZ/Ort

Widerrufsrecht

Ich weiß, daß ich diese Vereinbarung innerhalb einer Woche beim DMV-Verlag, Postfach 250, 3440 Eschwege, schriftlich widerrufen kann, wobei bereits die rechtzeitige Absendung meines Widerrufschreibens zur Fristwahrung ausreicht. Ich bestätige das durch meine zweite Unterschrift.

Datum 2. Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Bitte ausreichend frankieren

DMV-Verlag
DOS International
Postfach 250

3440 Eschwege

1. Abonnieren

Jeden Monat neu, vermitteln Ihnen die DOS International und DOS EXTRA wichtige Informationen zu Ihrem Personal-Computer. Nutzen Sie dieses Wissen mit einem Abonnement zu Ihrem ganz persönlichen

- **Preisvorteil**
- **Informationsvorsprung**
- **Zustellkomfort**

Zur Erinnerung:
Mein persönliches Abonnement wurde bestellt am

Widerrufsrecht

Ich weiß, daß ich diese Vereinbarung innerhalb einer Woche beim DMV-Verlag, Postfach 250, 3440 Eschwege, schriftlich widerrufen kann, wobei bereits die rechtzeitige Absendung des Widerrufschreibens zur Fristwahrung ausreicht.

DOS INTERNATIONAL

Einzelhefte

Ja, ich mache von Ihrem Angebot Gebrauch und bestelle die umseitig ausgewählten Produkte. Bitte liefern Sie an folgende Anschrift:

Name

Straße/Postfach

PLZ/ORT

☐ Den Betrag bezahle ich mit dem beigefügten Verrechnungsscheck.

☐ Ich bitte um Lieferung per Nachnahme zuzüglich der Nachnahmegebühr (nur innerhalb der BRD).

Datum Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Bitte ausreichend frankieren

DMV-Verlag
DOS International
Postfach 250

3440 Eschwege

2. Sammeln

DOS-International-Leser sind immer top-informiert. Daher sollte Ihnen kein Heft oder Sonderheft in Ihrer Sammlung fehlen. Das Eintippen von Listings und Programmen ersparen Sie sich durch unseren günstigen Disketten-Service. Immer gut beraten sind Sie mit DOS International

- **Einzelheften**
- **Sonderheften**
- **Disketten-Service**

Zur Erinnerung:
Meine Hefte und Disketten wurden bestellt am

DOS INTERNATIONAL

Abonnement

Rechnungsanschrift

Vorname, Name

Straße, Nr./Postfach

PLZ/Ort

Widerrufsrecht

Ich weiß, daß ich diese Vereinbarung innerhalb einer Woche beim DMV-Verlag, Postfach 250, 3440 Eschwege, schriftlich widerrufen kann, wobei bereits die rechtzeitige Absendung meines Widerrufschreibens zur Fristwahrung ausreicht. Ich bestätige das durch meine zweite Unterschrift.

Datum 2. Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Bitte ausreichend frankieren

DMV-Verlag
DOS International
Postfach 250

3440 Eschwege

3. Schenken

Bei jedem Anlaß ist ein Abonnement ein Geschenk, das ankommt und Ihrem Wunsch nach »gutem Erfolg« Ausdruck verleiht. Sie brauchen nur im Feld »Lieferanschrift« die Adresse des Beschenkten einzutragen.

- **DOS International**
- **DOS EXTRA**

Zur Erinnerung:
Mein Geschenk-Abonnement wurde bestellt am

Widerrufsrecht

Ich weiß, daß ich diese Vereinbarung innerhalb einer Woche beim DMV-Verlag, Postfach 250, 3440 Eschwege, schriftlich widerrufen kann, wobei bereits die rechtzeitige Absendung des Widerrufschreibens zur Fristwahrung ausreicht.

Römische Zahlen: MCMXCI

Die meisten Menschen, vor allem aber jene, die sich mit einem Computer näher beschäftigen, haben mit vielen verschiedenartigen Zahlensystemen zu tun. Ein so altes Zahlensystem wie das der Römer gerät dabei leicht in Vergessenheit. Manchmal konfrontiert Sie jedoch der Alltag, beispielsweise an Portalen, in Museen oder im Filmnachspann, mit diesen Zahlen. Um zu zeigen, wie leicht das römische Zahlensystem aufgebaut ist, und wie einfach die Umwandlung von römischen Dezimalzahlen erfolgen kann, finden Sie die Programme »r_zahl.pas« (Listing 5) und »zahl_r.pas« (Listing 6). »r_zahl.pas« wandelt eine vorgegebene römische Zahl in eine Dezimalzahl um, während »zahl_r.pas« eine Dezimalzahl in eine römische Zahl umwandelt. Die Werte der römischen Zahlen entnehmen Sie Tabelle 3.

Römische Zahlen haben im Gegensatz zum Dezimalsystem immer den gleichen Wert, egal an welcher Stelle sie stehen. Für den Wert einer Zahl ist nur die Stellung der einzelnen Zahlen zueinander wichtig. Steht eine größere Zahl links von einer kleineren, so werden deren Werte addiert. »XI« entspricht also dem Wert 11. Steht eine kleinere Zahl links von einer größeren Zahl, so wird die kleinere Zahl von der größeren abgezogen. »IX« entspricht infolgedessen dem Wert 9. Diese Art der Zahlenstellung zueinander wird nur angewandt, wenn die zu bearbeitende Zahl eine 9 oder eine 4 ist, da 9 als 10 – 1 (IX) und 4 als 5 – 1 (IV) dargestellt wird. Die Zahlen 40 , 400 , 90 , 900 schrieben die Römer also als »XL«, »CD«, »XC«, »CM«. Sind zwei nebeneinanderstehende Zahlen gleich, so werden sie addiert. »II« ergibt also 2. Bei der Umwandlung einer römischen Zahl in eine Dezimalzahl rechnen Sie immer von rechts nach links. Beispielsweise entspricht »MCMXCI« der Zahl 1991, da

```
I = 1
XC = 90
CM = 900
M = 1000
```

unsere Jahreszahl ergibt: 1000 + 900 + 90 + 1 = 1991.

(Michael Humpl/et)

Römische Zahl	Dezimalzahl
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

```
[DP3]    1: {Programm: r_zahl.pas
[656]    2:  Funktion: Römische -> Dezimalzahlen
[SZ5]    3:  Sprache : Turbo Pascal ab Version 5.Ø
[EA3]    4:  Autor: Michael Humpl
[TX4]    5:  (c)1991 DMV Widuch GmbH & Co. KG}
[ZF3]    6: var eingabe: string;
[SY1]    7:     zaehl: Byte;
[B65]    8:     zahl,stelle,speicher: Longint;
[8X2]    9: procedure fehler;
[MUØ]   1Ø: begin
[5E4]   11:   WriteLn('Der Buchstabe -> ',
[DP2]   12:   eingabe[zaehl],
[5C6]   13:   '<- ist nicht als Zahlenwert bekannt!');
[7A5]   14:   WriteLn('Programm abgebrochen');
[P41]   15:   Halt(Ø);
[ZNØ]   16: end;
[MUØ]   17: begin
[PF1]   18:   zahl := Ø;
[L14]   19:   speicher := maxlongint;
[K15]   2Ø:   WriteLn('Bitte geben Sie die');
[225]   21:   Write(' römische Zahl ein : ');
[DH2]   22:   ReadLn(eingabe);
[XF1]   23:   WriteLn;
[MH5]   24:   for zaehl := 1 to Length(eingabe) do
[MUØ]   25:   begin
[455]   26:     case UpCase(eingabe[zaehl]) of
[GU2]   27:       'M': stelle := 1ØØØ;
[AT2]   28:       'D': stelle := 5ØØ;
[3T2]   29:       'C': stelle := 1ØØ;
[RL2]   3Ø:       'L': stelle := 5Ø;
[RM2]   31:       'X': stelle := 1Ø;
[UF2]   32:       'V': stelle := 5;
[FE2]   33:       'I': stelle := 1;
[R22]   34:       else fehler;
[ZNØ]   35:     end;
[Ø14]   36:     if stelle > speicher then
[HM4]   37:     Inc(zahl,stelle-2*speicher)
[ZRØ]   38:     else
[8X2]   39:     Inc(zahl,stelle);
[E83]   4Ø:     speicher := stelle;
[ZNØ]   41:   end;
[GB6]   42:   WriteLn('Die römische Zahl ',eingabe,
[625]   43:           ' entspricht der Dezimalzahl: ',
[6ZØ]   44:           zahl);
[KNØ]   45: end.
```

Listing 5. »r_zahl.pas« wandelt römische Zahlen in Dezimalzahlen um

```
[KN3]    1: {Programm: zahl_r.pas
[E55]    2:  Funktion: Dezimahl -> röm. Zahl
[SZ5]    3:  Sprache : Turbo Pascal ab Version 5.Ø
[XR3]    4:  Autor:    Michael Humpl
[TX4]    5:  (c)1991 DMV Widuch GmbH & Co. KG)
[FH3]    6: var ausgabe: string;
[MB6]    7:     zahl,stelle,speicher,hilfs: Longint;
[MUØ]    8:     begin
[M91]    9:       zahl:= Ø;
[8Ø2]   1Ø:       speicher:= Ø;
[XX1]   11:       ausgabe:= '';
[MA4]   12:       Write('Bitte geben Sie die '+
[DA3]   13:             'Dezimalzahl ein : ');
[Ø32]   14:       ReadLn(zahl);
[252]   15:       hilfs:= zahl;
[XF1]   16:       WriteLn;
[TW3]   17:       speicher:= zahl DIV 1ØØØ;
[BK4]   18:       for stelle:= 1 to speicher do
[HC3]   19:       ausgabe:= ausgabe+'M';
[Y14]   2Ø:       zahl:= zahl-speicher*1ØØØ;
[SK2]   21:       if zahl>=9ØØ then
[MUØ]   22:       begin
[3M3]   23:         ausgabe:= ausgabe+'CM';
[WC2]   24:         zahl:= zahl-9ØØ;
[ZNØ]   25:       end;
[CK2]   26:       if zahl>=5ØØ then
[MUØ]   27:       begin
[8C3]   28:         ausgabe:= ausgabe+'D';
[MC2]   29:         zahl:= zahl-5ØØ;
[ZNØ]   3Ø:       end;
[9K2]   31:       if zahl>=4ØØ then
[MUØ]   32:       begin
[HL3]   33:         ausgabe:= ausgabe+'CD';
[KC2]   34:         zahl:= zahl-4ØØ;
[ZNØ]   35:       end;
[ØK2]   36:       if zahl>=1ØØ then
[MUØ]   37:       begin
[ET3]   38:         speicher:= zahl DIV 1ØØ;
[BK4]   39:         for stelle:= 1 to speicher do
[7C3]   4Ø:         ausgabe:= ausgabe+'C';
[3X3]   41:         zahl:= zahl-speicher*1ØØ;
[ZNØ]   42:       end;
[DF2]   43:       if zahl>=9Ø then
[MUØ]   44:       begin
[4M3]   45:         ausgabe:= ausgabe+'XC';
[DC2]   46:         zahl:= zahl-9Ø;
[ZNØ]   47:       end;
[1F2]   48:       if zahl>=5Ø then
[MUØ]   49:       begin
[GC3]   5Ø:         ausgabe:= ausgabe+'L';
[5C2]   51:         zahl:= zahl-5Ø;
[ZNØ]   52:       end;
[XG2]   53:       if zahl>= 4Ø then
[MUØ]   54:       begin
[PM3]   55:         ausgabe:= ausgabe+'XL';
[3C2]   56:         zahl:= zahl-4Ø;
[ZNØ]   57:       end;
[HG2]   58:       if zahl>= 1Ø then
[MUØ]   59:       begin
[ZS3]   6Ø:         speicher:= zahl DIV 1Ø;
[BK4]   61:         for stelle:= 1 to speicher do
[WC3]   62:         ausgabe:= ausgabe+'X';
[ST3]   63:         zahl:= zahl-speicher*1Ø;
[ZNØ]   64:       end;
[6E2]   65:       if zahl>= 9 then
[MUØ]   66:       begin
[ZM3]   67:         ausgabe:= ausgabe+'IX';
[262]   68:         zahl:= zahl-9;
[ZNØ]   69:       end;
[PD2]   7Ø:       if zahl>= 5 then
[MUØ]   71:       begin
[UC3]   72:         ausgabe:= ausgabe+'V';
[U52]   73:         zahl:= zahl-5;
[ZNØ]   74:       end;
```

Listing 6. »zahl_r.pas« gibt römische Zahlen in arabischen Ziffern aus

```
[KD2]   75:        if zahl>= 4 then
[MUØ]   76:        begin
[VM3]   77:          ausgabe:= ausgabe+'IV';
[S52]   78:          zahl:= zahl-4;
[ZNØ]   79:        end;
[6D2]   8Ø:        if zahl>= 1 then
[MUØ]   81:        begin
[LW3]   82:          for stelle:= 1 to zahl do
[DC3]   83:          ausgabe:= ausgabe+'I';
[CE3]   84:          zahl:= zahl-speicher;
[ZNØ]   85:        end;
[TB5]   86:        WriteLn('Die Dezimalzahl ',hilfs,
[KX5]   87:        ' entspricht der römischen Zahl : '
[3L1]   88:        ,ausgabe);
[KNØ]   89:      end.
```

Listing 6. »zahl__r.pas« gibt römische Zahlen in arabischen Ziffern aus

Komfortabel kopieren mit einem Laufwerk

Endlich hat das ewige Diskettenwechseln ein Ende. Von nun an kopiert der versierte PC-Anwender seine 1,2-MByte-Disketten nur noch mit »hdcopy.c« (Listing 7). Hdcopy überträgt alle Dateien der Quelldiskette auf Laufwerk C und anschließend auf die Zieldiskette. Die Festplatte wird selbstverständlich nach getaner Arbeit wieder gesäubert. Wer also eine Festplatte und nur ein Diskettenlaufwerk sein eigen nennt, braucht von nun an nicht mehr sechsmal die Diskette zu wechseln (wie mit »diskcopy«), sondern – dank Hdcopy – nur noch ein einziges Mal.

Hdcopy benutzt eine Funktion des VGA-BIOS, welche den Bildschirmaufbau verhindert. Das Programm ist aber auch mit einer Hercules-Karte lauffähig: in diesem Falle wird die unerwünschte Meldung nicht unterdrückt. Sie kompilieren Hdcopy mit Turbo C der Version 1.01.

(H. Straßburg/T. Riethmüller/et)

```
[GA3]    1: /*Programm: hdcopy.c
[PK5]    2:   Funktion: komfortables Diskcopy
[D53]    3:   Sprache : Turbo C ++
[PY3]    4:   Autor:    H. Straßburg
[T15]    5:   (c)1991 DMV Widuch GmbH & Co. KG */
[L13]    6: #include <stdio.h>
[2Ø3]    7: #include <conio.h>
[1L2]    8: #include <dir.h>
[9L2]    9: #include <dos.h>
[VG2]   1Ø: #include <io.h>
[NC2]   11: int main(void)
[6Z1]   12: { int lw,a,n;
[M32]   13:   FILE *stream;
[2T2]   14:   struct REGPACK r;
[ØE1]   15:   clrscr();
[X32]   16:   lw=getdisk();
[LX1]   17:   setdisk(2);
[V52]   18:   chdir("c:\\");
[EB6]   19:   printf("BITTE QUELLDISKETTE EINLEGEN"
[S44]   2Ø:           " UND TASTE DRÜCKEN\n");
[R61]   21:   getch();
[F22]   22:   _creat("Y",Ø);
[XH3]   23:   stream=fopen("Y","w");
[WX3]   24:   fprintf(stream,"%s","Y");
[GH2]   25:   fclose(stream);
[H62]   26:   r.r_ax=Øx12Ø1;
[N62]   27:   r.r_bx=ØxØØ36;
[Ø32]   28:   intr(Øx1Ø,&r);
[RX4]   29:   system("backup a: c:/s <Y >nul");
[ØE1]   3Ø:   clrscr();
[F62]   31:   r.r_ax=Øx12ØØ;
[N62]   32:   r.r_bx=ØxØØ36;
[Ø32]   33:   intr(Øx1Ø,&r);
[W36]   34:   printf("BITTE ZIELDISKETTE EINLEGEN"
[S44]   35:           " UND TASTE DRÜCKEN\n");
[FY4]   36:   system("restore c: a:/s>nul");
[F16]   37:   a=_chmod("c:\\backup\\backup.ØØ1",Ø);
[EH2]   38:   n=a^FA_RDONLY;
[8X5]   39:   _chmod("c:\\backup\\backup.ØØ1",1,n);
[M66]   4Ø:   _chmod("c:\\backup\\control.ØØ1",1,n);
[943]   41:   chdir("c:\\backup");
[D33]   42:   remove("backup.ØØ1");
[UF3]   43:   remove("control.ØØ1");
[V52]   44:   chdir("c:\\");
[AT1]   45:   remove("Y");
[VB2]   46:   rmdir("backup");
[B56]   47:   printf("\nKOPIERVORGANG BEENDET\n");
[432]   48:   setdisk(lw);
[U7Ø]   49: }
```

Listing 7. »hdcopy.c« kopiert komfortabel mit nur einem Laufwerk

Prüfsumme als Schutz im Programm

»chk__vir.pas« (Listing 8) kompilieren Sie mit Turbo Pascal ab 5.5 zu einer Unit, die Sie in jedes Turbo-Pascal-Programm einbinden können. Diese Unit vergleicht nicht nur die Dateigröße, sondern legt auch eine Prüfsumme des jeweiligen Programms an, die Sie im Programmcode ablegen. So wie Sie den Quellcode jetzt vor sich liegen haben, finden Sie die Programmzeilen 35 und 36

```
kenz:='@@@@@@@@@@'
comp:='.........'
```

Jede dieser Zeichenketten muß zehn Zeichen lang sein. Der erste Ausdruck dient der Kennzeichnung, um beim Bilden der Prüfsumme diese nicht selbst mitzurechnen. Der zweite Ausdruck enthält momentan nur Leerstellen. Sie können den ersten Ausdruck natürlich nach Belieben verändern. Die Zeichenfolge muß auf jeden Fall im gesamten Programmcode einmalig sein. Allerdings ist diese Unit nur einzubinden, wenn die Entwicklung des Programms schon abgeschlossen ist. Es dürfen auch keine Debuginformationen im ausführbaren Programm enthalten sein. Schalten Sie also »Debuginformation« im »Optionsmenü« aus. Führen Sie nun folgende Schritte durch:

- Kompilieren Sie »chk__vir.pas« ganz normal ohne irgendeinen Parameter. Sie erhalten »chk__vir.exe«.
- Kompilieren Sie anschließend den gleichen Quellcode mit der Angabe des Parameters »u« für Unit im Menü »Options Compiler« in der Zeile »Conditional Defines«. Sie erhalten so die Unit »chk__vir.tpu.«
- Binden Sie »chk__vir« über eine Uses-Anweisung in ein beliebiges Pascal-Programm ein, und kompilieren Sie dieses. Sie erhalten zum Beispiel eine Datei »test.exe«.
- Rufen Sie »chk__vir« mit dem Programm »test.exe« und seinem vollständigen Pfad als Parameter auf, also zum Beispiel:

```
chk_vir c:\prog\turbo\exe\test.exe
```

»chk__vir« gibt Ihnen eine mehrstellige Zahl aus. Diese fügen Sie unter den Hochkommata in den Quellcode der Zeile 36 »comp: = ''« ein. Die Zahl tragen Sie rechtsbündig ein. Der Text muß immer noch zehn Zeichen lang sein. Also beispielsweise:

```
comp:= '1234567';
```

Kompilieren Sie »chk__vir« mit dem Parameter »u« erneut und anschließend das Programm »test.pas«. Test können Sie jetzt ganz normal starten. Sobald Sie aber nur ein Byte in »test.exe« ändern, bricht es zum Beispiel mit der Fehlermeldung

```
Runtime Error 33
```

ab. Falls das Programm trotz richtiger Durchführung aller Schritte mit einem Laufzeitfehler abbricht, belegen Sie die Variable »B« mit einem anderen Wert und versuchen es von neuem. Daß ein wiederholter Installationslauf notwendig wird, ist sehr unwahrscheinlich.

(Robert Ortner/et)

```
[BX3]    1: {Programm: chk_vir.pas
[ZM7]    2:  Funktion: Prüfsummenverfahren: Virenschutz
[2Ø6]    3:  Sprache : Turbo Pascal ab Version 5.5
[WC3]    4:  Autor: Robert Ortner
[P35]    5:  (c)1991 DMV Widuch GmbH & Co. KG }
[WV1]    6: {$IFDEF u}
[C92]    7: unit chk_vir;
[SK1]    8: interface
[4L2]    9: implementation
[WD1]   1Ø: {$ENDIF}
[VG1]   11: uses Dos;
[R72]   12: const B = 5ØØØØ;
[WW1]   13: var f: file;
[TW1]   14:     P: PathStr;
[RL1]   15:     D: DirStr;
[KV1]   16:     N: NameStr;
[YM1]   17:     E: ExtStr;
```

Listing 8. »chk__vir.pas« belegt Pascal-Programme mit einer Prüfsumme, die sie vor Veränderung schützt

```
[7H3]   18:     chk,cmp,fs: Longint;
[432]   19:     NR,j,i: Word;
[674]   2Ø:     buf: array[1..B] of Byte;
[GP3]   21:     kenz,comp: string[1Ø];
[LN2]   22:     kenzp: ^string;
[EP2]   23:     Info: SearchRec;
[MUØ]   24:     begin
[5Y5]   25:       {$IFDEF u} P:= ParamStr(Ø); {$ENDIF}
[276]   26:       {$IFNDEF u} P:= ParamStr(1);{$ENDIF}
[UL2]   27:       FSplit (P,D,N,E);
[4K1]   28:       Dec(D[Ø]);
[2C1]   29:       ChDir(D);
[3H4]   3Ø:       FindFirst(P,Anyfile,Info);
[FL2]   31:       fs:= Info.Size;
[452]   32:       Assign(f,N+E);
[HT1]   33:       Reset(f,1);
[U51]   34:       chk:=Ø;
[ØZ2]   35:       kenz:= '@@@@@@@@@@';
[Ø62]   36:       comp:= '          ';
[SL2]   37:       Val(comp,cmp,j);
[WPØ]   38:       j:=Ø;
[P31]   39:       repeat
[G85]   4Ø:         BlockRead(f,buf,SizeOf(buf),NR);
[UPØ]   41:         i:=Ø;
[B51]   42:         Inc(j);
[BK3]   43:         while (i<B) AND (i<fs) do
[MUØ]   44:         begin
[951]   45:           Inc(i);
[TV2]   46:           kenzp:= @buf[i];
[HA2]   47:           comp:=kenzp^;
[V84]   48:           if comp=kenz then Inc(i,22)
[TZ3]   49:           else Inc(chk,buf[i]*i);
[ZNØ]   5Ø:         end;
[NA3]   51:         if j*B>=fs then NR:=Ø;
[DY1]   52:       until (NR=Ø);
[KD1]   53:       Close(f);
[WV1]   54:       {$IFDEF u}
[SV4]   55:       if cmp<>chk then RunError(33);
[WD1]   56:       {$ENDIF}
[M96]   57:       {$IFNDEF u} WriteLn(chk); {$ENDIF}
[KNØ]   58:     end.
```

Listing 8. »chk_vir.pas« belegt Pascal-Programme mit einer Prüfsumme, die sie vor Veränderung schützt

Die Konstanten »E« und »Pi«

Die beiden wichtigsten Konstanten in der Mathematik sind die Zahlen »E« und »Pi«. Die Eulersche Zahl

E=2,7182818...

ist die Basis des natürlichen Logarithmus und spielt insbesondere bei Wachstumsprozessen eine wichtige Rolle. Die Zahl

π=3,1415926...

ist die sogenannte Kreiszahl: »Pi« beinhaltet das Verhältnis von Umfang und Durchmesser eines Kreises. Daneben gilt für den Umfang »A« eines Kreises mit dem Radius »r« die Beziehung

A=2*π*r

Eine bemerkenswert genaue Berechnung der Zahl »Pi« gelang im Jahr 1610 Ludolph van Ceulen, der »Pi« auf 35 Dezimalstellen genau berechnete und zu dessen Ehren »Pi« auch »Ludolphsche Zahl« genannt wird. Spätere genauere Berechnungen sind wohl teilweise aus der Frage entstanden, ob die Dezimalbruchentwicklung von »Pi« einmal abbricht. Allerdings wies J.H. Lambert im Jahre 1761 nach, daß »Pi« eine irrationale Zahl ist. Daraus folgt, daß die Dezimalbruchentwicklung von »Pi« nicht abbricht und auch nicht periodisch ist. Lindemann zeigte im Jahr 1882, daß »Pi« eine transzendente Zahl ist, das heißt, »Pi« ist nicht eine Lösung einer algebraischen Gleichung mit rationalen Koeffizienten. Dies bewies zugleich, daß das Problem der Quadratur des Kreises nicht lösbar ist, nichts und niemand kann also nur mit Zirkel und Lineal einen Kreis in ein flächengleiches Quadrat verwandeln. Auch für die Zahl »E« wies Hermite 1873 die Irrationalität und die Transzendenz nach. Insbesondere folgt, daß auch die Dezimalbruchentwicklung von »E« nicht abbricht und auch nicht periodisch ist. Der Berechnung der Zahl »E« liegt in »epi.pas« (Listing 9) die Taylor-Entwicklung der Exponentialfunktion zugrunde (Bild 2), die auch noch auf die Formel in Bild 3 zurückgreift.

$$ex=1+\frac{x}{1!}+\frac{x2}{2!}+\frac{x3}{3!}+\frac{x4}{4!}+\frac{x5}{5!}\ldots$$

Bild 2. Die Berechnung der Eulerschen Zahl »E« beruht auf der Taylorschen Reihe

$$\frac{1}{(n+1)!}=\frac{1}{n!}:(n+1)$$

Bild 3. Den folgenden Summanden bei der Berechnung von »E« gewinnen Sie jeweils durch Division aus dem vorhergehenden

Die Berechnung der Zahl »Pi« fußt in Epi auf der Taylor-Entwicklung für die Arcus-Tangens-Funktion (Bild 4) und auf der Formel von Machin (Bild 5). Epi soll zeigen, daß sich die Konstanten »E« und »Pi« auch schon mit einem PC in vertretbarer Rechenzeit auf bis zu 10000 Dezimalstellen (Bild 6) berechnen lassen. Wenn Sie den Quelltext leicht verändern, können Sie sogar etwa 32000 Dezimalstellen erreichen. Hierzu ändern Sie einfach eine Konstantendeklaration:

```
MaxStellenzahl=32000
```

Allerdings erhöht sich dann auch die Rechenzeit. Mit leistungsfähigen Großrechnern hat man insbesondere »Pi« schon auf etliche Millionen Dezimalstellen genau berechnet. Nun zu einigen Einzelheiten des Programms. Grundlegende Datenstruktur ist der Typ »Langzahl«, ein Feld mit maximal 2511 Feldkomponenten. Die Feldkomponenten enthalten ganze Zahlen in den Grenzen 0 und 9999 (einschließlich). Die Dezimalbruchentwicklungen von »E« und »Pi« werden schließlich als Langzahlen berechnet und ausgegeben.

Die Prozeduren »scrmode«, »cursoroff« und »cursoron« machen den Cursor unsichtbar und wieder sichtbar. Die Prozeduren »Fullscreen« und »Fenster« dienen der grafischen Gestaltung des Programms. Die Prozedur »Zeit« dient zur Ausgabe der jeweils benötigten Rechenzeit. »Loeschen« ist dazu da, sämtliche Komponenten einer Langzahl auf Null zu setzen. Die Prozedur »WriteLangzahl« dient der Ausgabe einer Langzahl auf dem Bildschirm oder mit Hilfe eines Druckers; dabei werden führende Nullen einzelner Feldkomponenten ergänzt. Die Prozeduren »Division«, »Addition« und »Subtraktion« bilden eine reduzierte Langzahl »Arithmetik«; inhaltlich orientieren sie sich an den Algorithmen für das schriftliche Dividieren, Addieren und Subtrahieren ganzer Zahlen.

Die Prozedur »e_Berechnung« enthält zunächst die Abfrage nach der gewünschten Stellenzahl. Wesentlicher Teil dieser Prozedur ist die Berechnung von »E« nach der Formel aus Bild 2. Dabei wird der folgende Summand jeweils durch Division aus dem vorhergehenden berechnet (Bild 3).

Mit der Prozedur »ArcusTangens« berechnen Sie Terme nach der Formel in Bild 4. Dazu wird die Taylor-Entwicklung für ArcusTangens in einen adäquaten Algorithmus umgesetzt. Die Prozedur »Pi_Berechnung« enthält eine Abfrage der gewünschten Stellenzahl und die Berechnung der Zahl »Pi« unter Rückgriff auf die Prozedur »ArcusTangens« (Formel von Machin, Bild 5).

$$\arctan x=x-\frac{x3}{3}+\frac{x5}{5}-\frac{x7}{7}+\frac{x9}{9}-\ldots$$

Bild 4. Die Berechnung der Zahl »Pi« fußt in »epi.pas« auf der Taylor-Entwicklung für die Arcustangens-Funktion

$$\pi=16*\arctan\frac{1}{5}-4*\arctan\frac{1}{239}$$

Bild 5. Auch mit der Formel von Machin berechnen Sie die Kreiszahl »Pi«

```
    2561 2521 6320 8628 6922 2103 2748 8921 8654 3648 0229 6780 7057 6561
    5144 6320 4692 7906 8212 0738 8377 8142 3356 2823 6089 6320 8068 2224
    6801 2248 2611 7718 5896 3814 0918 3903 6736 7222 0888 3215 1375 5600
    3727 9839 4004 1529 7002 8783 0766 7094 4474 5601 3455 6417 2543 7090
    6979 3961 2257 1429 8946 7154 3578 4687 8861 4445 8123 1459 3571 9849
    2252 8471 6050 4922 1242 4701 4121 4780 5734 5510 5008 0190 8699 6033
    0276 3478 7081 0817 5450 1193 0714 1223 3908 6639 3833 9529 4257 8690
    5076 4310 0638 3519 8343 8934 1596 1318 5434 7546 4955 6978 1038 2930
    9716 4651 4384 0700 7073 6041 1237 3599 8434 5225 1610 5070 2705 6235
    2660 1276 4848 3084 0761 1830 1385 2793 2054 2746 2865 4036 0367 4532
    8651 0570 6587 4882 2569 8157 9367 8976 6974 2205 7505 9683 4408 6973
    5020 1410 2067 2358 5020 0724 5225 6326 5134 1055 9240 1902 7421 6248
    4391 4035 9989 5353 9459 0944 0704 6912 0914 0938 7001 2645 6001 6237
    4288 0210 9276 4579 3106 5792 2955 2498 8727 5846 1012 6483 6999 8922
    5695 9688 1592 0560 0101 6552 5637 5678


Die Rechenzeit betrug   1 Std 50 Min 32.85 Sek .

Zur Rückkehr ins Menu <Eingabe> drücken !
```

Bild 6. Auf einem 80386-PC braucht »epi.pas« beinahe zwei Stunden, um die Kreiszahl »Pi« mit 9999 Nachkommastellen zu errechnen

Die Prozedur »Menu« gestattet die Auswahl, welche der beiden Konstanten »E« und »Pi« berechnet werden soll und ob eine zusätzliche Ausgabe auf einem Drucker erfolgen soll. Zum Abschluß folgen einige Angaben über den Zeitbedarf des Programms. Auf einem 80386-PC mit 25 MHz Taktfrequenz braucht das Programm zur Berechnung der Zahl »E« auf 10000 Stellen etwa 20 Minuten, während es zur Berechnung der Zahl »Pi« auf 10000 Stellen etwa 1 Stunde 52 Minuten benötigt (Bild 6). Die Routinen sind besonders dann sinnvoll, wenn Sie sie mit Berechnungs-Routinen kombinieren, die mit den Zahlen derselben Genauigkeit umgehen können. Schicken Sie die Ergebnisse Ihrer Anstrengungen an die DOS Redaktion. Für einen Beitrag unter Tips & Tricks sind sie sicher gut.

Literatur: D. Herrmann/M. Weber: Mathematik auf dem Commodore 64, IWT-Verlag, ISBN 3-88322-048/-5

(Dr. Raimund Petow/et)

```
[Ø33]    1: {Programm: epi.ori
[VK4]    2:  Funktion: Berechnet Pi und e
[EV5]    3:  Sprache: Turbo Pascal ab Version 5.Ø
[7B4]    4:  Autor:   Dr. Raimund Petow
[TX4]    5:  (c)1991 DMV Widuch GmbH & Co. KG}
[D52]    6: program EPI;
[TW2]    7: {$M 6552Ø,Ø,65536Ø}
[ZK5]    8: {Festlegung der Stackgröße}
[FN3]    9: uses Crt,Printer,Dos;
[1Z3]   1Ø: const Zehntausend=1ØØØØ;
[FG3]   11:       MaxStellenzahl=1ØØØØ;
[3V5]   12:       Feldlaenge=MaxStellenzahl DIV 4+1Ø;
[U46]   13:       {Absicherung gegen Rundungsfehler}
[TR4]   14: type BLOCK=Ø..Zehntausend-1;
[YS6]   15:      LANGZAHL=array[Ø..Feldlaenge] of BLOCK
        ;
[A74]   16: var x,Ergebnis: LANGZAHL;
[4W4]   17:     Stellenzahl,Blockzahl: Word;
[5W4]   18:     Std,Min,Sek,HSek: SHORTINT;
[PA3]   19:     Std1,Min1,Sek1,HSek1,
[ØF4]   2Ø:     Std2,Min2,Sek2,HSek2: Word;
[HK3]   21:     register: Registers;
[YØ4]   22: function scrmode: Byte;
[MUØ]   23: begin
[RS5]   24:   register.AH:=$ØF; Intr($1Ø,register);
[VM3]   25:   scrmode:=register.AL;
[Z23]   26: end; {of scrmode}
[AK3]   27: procedure cursoroff;
[MUØ]   28: begin
[KS3]   29:   with register do begin
[PV4]   3Ø:     if scrmode=7 then CX:=$3ØØØ else
[2T2]   31:     CX:=$2ØØØ; AX:=$Ø1ØØ;
[PN2]   32:   end; {of with}
[253]   33:   Intr($1Ø,register);
[XS3]   34: end; {of cursoroff}
[VF3]   35: procedure cursoron;
[MUØ]   36: begin
[KS3]   37:   with register do begin
[3Z4]   38:     if scrmode=7 then CX:=$ØBØC else
[2U2]   39:     CX:=$Ø6Ø7; AX:=$Ø1ØØ;
[PN2]   4Ø:   end; {of with}
[253]   41:   Intr($1Ø,register);
[YE3]   42: end; {of cursoron}
[YN3]   43: procedure Fullscreen;
[MUØ]   44: begin
[5V3]   45:   Window(1,1,8Ø,25); ClrScr;
[1N3]   46: end; {of Fullscreen}
[6Ø6]   47: procedure Fenster(x1,y1,x2,y2: Byte);
[P34]   48: const Eckeobenlinks=#2Ø1;
[DH3]   49:       Liniehorizontal=#2Ø5;
[A93]   5Ø:       Eckeobenrechts=#187;
[Ø73]   51:       Linievertikal=#186;
[MB3]   52:       Eckeuntenlinks=#2ØØ;
[KK3]   53:       Eckeuntenrechts=#188;
[DY1]   54:       var i: Word;
[MUØ]   55: begin
[U14]   56:   Window(x1-1,y1-1,x2+1,y2+1);
[XX3]   57:   ClrScr; Window(1,1,8Ø,25);
[SA6]   58:   GotoXY(x1-1,y1-1); Write(Eckeobenlinks);
[LV2]   59:   for i:=x1 to x2 do
[M44]   6Ø:   Write(Liniehorizontal);
[XU3]   61:   Write(Eckeobenrechts);
[AS3]   62:   for i:=y1 to y2 do begin
[S36]   63:     GotoXY(x1-1,i); Write(Linievertikal);
[M36]   64:     GotoXY(x2+1,i); Write(Linievertikal);
[ZNØ]   65:   end;
[TM6]   66:   GotoXY(x1-1,y2+1); Write(Eckeuntenlinks);
[LV2]   67:   for i:=x1 to x2 do
[M44]   68:   Write(Liniehorizontal);
[H24]   69:   Write(Eckeuntenrechts);
[P43]   7Ø:   Window(x1,y1,x2,y2);
[833]   71: end; {of Fenster}
[2T2]   72: procedure Zeit;
[1E3]   73: var Uebertrag: Byte;
[MUØ]   74: begin
[S65]   75:   Uebertrag:=Ø; HSek:=HSek2-HSek1;
[353]   76:   if(HSek<Ø)then begin
[GM4]   77:     HSek:=HSek+1ØØ; Uebertrag:=1;
[ZNØ]   78:   end;
[BG6]   79:   Sek:=Sek2-Sek1-Uebertrag; Uebertrag:=Ø;
[Z33]   8Ø:   if(Sek<Ø)then begin
[674]   81:     Sek:=Sek+6Ø; Uebertrag:=1;
[ZNØ]   82:   end;
[AG6]   83:   Min:=Min2-Min1-Uebertrag; Uebertrag:=Ø;
[A43]   84:   if(Min<Ø)then begin
[L74]   85:     Min:=Min+6Ø; Uebertrag:=1;
[ZNØ]   86:   end;
[KK6]   87:   Std:=Std2-Std1-Uebertrag; Uebertrag:=Ø;
[843]   88:   if(Std<Ø)then begin
[AZ1]   89:     Std:=Std+24;
[ZNØ]   9Ø:   end;
[FE2]   91: end;{of Zeit}
[126]   92: procedure Loeschen(Blockzahl: Word;
[AB3]   93:           var Hilf: LANGZAHL);
[DY1]   94: var i: Word;
[MUØ]   95: begin
[RY5]   96:   for i:=Ø to Blockzahl do Hilf[i]:=Ø;
[YB3]   97: end; {of Loeschen}
[YZ6]   98: procedure WriteLangzahl(Blockzahl: Word;
[RH4]   99:           x: LANGZAHL;Druck: Boolean);
[DY1]  1ØØ: var i: Word;
[XK5]  1Ø1: Langstring: array[Ø..Feldlaenge]
[PK3]  1Ø2:             of string[4]; z: Char;
[MUØ]  1Ø3: begin
[9Z4]  1Ø4:   for i:=1 to Blockzahl do begin
[8V4]  1Ø5:     Str(x[i]: Ø,Langstring[i]);
[GB4]  1Ø6:     case Langstring[i][Ø]of
[B76]  1Ø7: #3: Langstring[i]:='Ø'+Langstring[i];
[FF6]  1Ø8: #2: Langstring[i]:='ØØ'+Langstring[i];
[BN6]  1Ø9: #1: Langstring[i]:='ØØØ'+Langstring[i];
[L34]  11Ø: #Ø: Langstring[i]:='ØØØØ';
[ED2]  111:     end;{of case}
[V44]  112:     Write(Langstring[i]: 5);
[MZ2]  113:     if(i>1)AND(i MOD 14=Ø)
[ØW2]  114:     then Write('       ');
[S53]  115:     if(i>1)AND(i MOD 126=Ø)
[7U5]  116:     then repeat z:=ReadKey until z=#13;
[YX3]  117:     if Druck=True then begin
[AU4]  118:       Write(lst,Langstring[i]: 5);
[MZ2]  119:       if(i>1)AND(i MOD 14=Ø)
[FR3]  12Ø:       then Write(lst,'          ');
[K92]  121:     end;{of if}
[S82]  122:   end;{of for}
[B13]  123:   if Druck=True then
[5X4]  124:   for i:=1 to 3 do WriteLn(lst);
[424]  125:   for i:=1 to 3 do WriteLn;
[N85]  126:   Write('  Die Rechenzeit betrug ',
[AH5]  127:   Std: 3,' Std',Min: 3,' Min',Sek: 3, '.');
[PA6]  128:   Write(HSek,' Sek .'); WriteLn;  WriteLn;
[DK5]  129:   Write('  Zur Rückkehr ins Menu');
[7M4]  13Ø:   Write(' <Eingabe> drücken !');
[D54]  131: end;{of WriteLangzahl}
[C36]  132: procedure Division(Zahl1: LANGZAHL;
[8T5]  133:           Teiler: Longint; Blockzahl: Word;
[W24]  134:           var Ergebnis: LANGZAHL;
[6K2]  135:           var z: Boolean);
[SF6]  136: var Rest,i,Divisor,Quotient: Longint;
[MUØ]  137: begin
[TX2]  138:   Rest:=Ø; z:=False;
[8Z4]  139:   for i:=Ø to Blockzahl do begin
[DD6]  14Ø:     Divisor:=Rest*Zehntausend+Zahl1[i];
[MØ5]  141:     Quotient:=Divisor DIV Teiler;
[F84]  142:     Rest:=Divisor MOD Teiler;
[WF3]  143:     z:=z OR(Quotient<>Ø);
[L24]  144:     Ergebnis[i]:=Quotient;
[CC2]  145:   end; {of for}
[DV3]  146:   Ergebnis[Blockzahl]:=
```

```
[LU3]  147:    Round(Divisor/Teiler);
[9C3]  148: end; {of Division}
[ØB6]  149: procedure Addition(Summand1,Summand2:
[4D4]  15Ø:                LANGZAHL; Blockzahl: Word;
[F24]  151:                var Ergebnis: LANGZAHL);
[WT4]  152: var Uebertrag,i,Summe: Word;
[MUØ]  153: begin
[562]  154:    Uebertrag:=Ø;
[5N5]  155:    for i:=Blockzahl downto Ø do begin
[A17]  156:    Summe:=Summand1[i]+Summand2[i]+Uebertrag;
[562]  157:      Uebertrag:=Ø;
[LF5]  158:      if(Summe>=Zehntausend)then begin
[BE4]  159:        Summe:=Summe-Zehntausend;
[G22]  16Ø:        Uebertrag:=1
[252]  161:      end; {of if}
[2C3]  162:    Ergebnis[i]:=Summe
[X72]  163:    end {of for}
[XA3]  164: end; {of Addition}
[X97]  165: procedure Subtraktion(Minuend,Subtrahend:
[1D4]  166:                LANGZAHL;Blockzahl: Word;
[F24]  167:                var Ergebnis: LANGZAHL);
[2Y6]  168: var Uebertrag: Byte; Differenz: Integer;
[P71]  169: i: Word;
[MUØ]  17Ø: begin
[562]  171:    Uebertrag:=Ø;
[5N5]  172:    for i:=Blockzahl downto Ø do begin
[KT3]  173:      Differenz:=Minuend[i]
[TB4]  174:      -Subtrahend[i]-Uebertrag;
[562]  175:      Uebertrag:=Ø;
[TØ4]  176:      if(Differenz<Ø)then begin
[VS5]  177:        Differenz:=Differenz+Zehntausend;
[G22]  178:        Uebertrag:=1
[252]  179:      end; {of if}
[724]  18Ø:      Ergebnis[i]:=Differenz
[X72]  181:    end {of for}
[NX3]  182: end; {of Subtraktion}
[DM6]  183: procedure e_Berechnung(Druck: Boolean);
[UU6]  184: var Teiler: Longint; Weiter,z: Boolean;
[1T2]  185: i: Byte; ch: Char;
[9Y4]  186: begin ClrScr; Fenster(2,2,78,23);
[BH5]  187:    WriteLn('  Berechnung der Zahl e');
[XF1]  188:    WriteLn;
[KH6]  189:    Write('  Anzahl der Nachkommastellen');
[PE4]  19Ø:    Write(' (bitte eingeben) = ');
[DXØ]  191:    {$I-}
[P31]  192:    repeat
[TZ1]  193:      GotoXY(5Ø,3);
[N74]  194:      for i:=1 to 2Ø do Write(' ');
[A45]  195:      GotoXY(5Ø,3); Read(Stellenzahl);
[1G6]  196:    until(IOResult=Ø)AND(Ø<Stellenzahl)AND
[5A5]  197:    (Stellenzahl<=MaxStellenzahl);
[BXØ]  198:    {$I+}
[XF1]  199:    WriteLn;
[LU4]  2ØØ:    Write('Zur Fortsetzung der ');
[BZ4]  2Ø1:    Write('Ausgabe Eingabe-Taste ');
[PY4]  2Ø2:    WriteLn('drücken !'); WriteLn;
[AM5]  2Ø3:    Blockzahl:=(Stellenzahl DIV 4)+1Ø;
[FP4]  2Ø4:    if Blockzahl>Feldlaenge then
[CT3]  2Ø5:       Blockzahl:=Feldlaenge;
[YX3]  2Ø6:    if Druck=True then begin
[Y23]  2Ø7:      repeat GotoXY(1,9);
[AS5]  2Ø8:        Write(' Ist der Drucker bereit?');
[T82]  2Ø9:        Write('(j/n)');
[222]  21Ø:        ch:=ReadKey;
[SA6]  211:      until ch IN['j','J']; WriteLn; WriteLn;
[252]  212:    end; {of if}
[R24]  213:    WriteLn('  e = '); WriteLn;
[512]  214:    Write('   2.');
[YX3]  215:    if Druck=True then begin
[ØH4]  216:      Write(lst,'Berechnung der ');
[1S5]  217:      Write(lst,'Zahl e auf Stellenzahl');
[HL5]  218:      WriteLn(lst,' Nachkommastellen');
[AB2]  219:      WriteLn(lst);
[7G5]  22Ø:      WriteLn(lst,'e = '); WriteLn(lst);
[1K2]  221:      Write(lst,'  2.');
[ZNØ]  222:    end;
[RX4]  223:    GetTime(Std1,Min1,Sek1,HSek1);
[BZ4]  224:    Loeschen(Blockzahl,Ergebnis);
[GP3]  225:    Loeschen(Blockzahl,x);
[845]  226:    x[Ø]:=1; Ergebnis:=x; Teiler:=1;
[P31]  227:    repeat
[GH6]  228:      Division(x,Teiler,Blockzahl,x,Weiter);
[W65]  229:      Addition(Ergebnis,x,Blockzahl,
[9P1]  23Ø:      Ergebnis);
[6U2]  231:      Teiler:=Teiler+1;
[ZX2]  232:    until NOT Weiter;
[8W5]  233:    GetTime(Std2,Min2,Sek2,HSek2); Zeit;
[PØ6]  234:    WriteLangzahl(Blockzahl-9,Ergebnis,
[GB4]  235:         Druck); WriteLn; WriteLn;
[P31]  236:    repeat
[GZ3]  237:      ch:=ReadKey until ch=#13;
[GF3]  238:    Fullscreen; ClrScr;
[R54]  239: end; {of e_Berechnung}
[Z66]  24Ø: procedure ArcusTangens(Koeffizient,
[PD4]  241:                Kehrwert,Blockzahl: Word;
[F24]  242:                var Ergebnis: LANGZAHL);
[RNØ]  243: var
[SG6]  244: Hilf1,Hilf2: LANGZAHL; Exponent,i: Word;
[ER7]  245: Teiler: Longint; Vorzeichen,Weiter:Boolean;
[MUØ]  246: begin
[ZB4]  247:    Loeschen(Blockzahl,Hilf1);
```

```
[ØC4]  248:    Loeschen(Blockzahl,Hilf2);
[BZ4]  249:    Loeschen(Blockzahl,Ergebnis);
[YS3]  25Ø:    Hilf1[Ø]:=Koeffizient;
[HX5]  251:    Division(Hilf1,Kehrwert,Blockzahl,
[1B2]  252:    Hilf2,Weiter);
[V85]  253:    Exponent:=1; Vorzeichen:=True;
[NL4]  254:    Teiler:=Kehrwert*Kehrwert;
[P31]  255:    repeat
[BØ4]  256:      for i:=Ø to Blockzahl do
[BC3]  257:      Hilf1[i]:=Hilf2[i];
[BY5]  258:      Division(Hilf2,Exponent,Blockzahl,
[1B2]  259:               Hilf2,Weiter);
[323]  26Ø:      if Vorzeichen then
[RS5]  261:      Addition(Ergebnis,Hilf2,Blockzahl,
[DA2]  262:      Ergebnis) else
[SA6]  263:      Subtraktion(Ergebnis,Hilf2,Blockzahl,
[5E5]  264:      Ergebnis); Exponent:=Exponent+2;
[UY4]  265:      Vorzeichen:=NOT Vorzeichen;
[3E5]  266:      Division(Hilf1,Teiler,Blockzahl,
[1B2]  267:      Hilf2,Weiter);
[433]  268:    until(NOT Weiter);
[454]  269: end; {of ArcusTangens}
[DV6]  27Ø: procedure pi_Berechnung(Druck: Boolean);
[SF3]  271: var i: Byte; ch: Char;
[MUØ]  272: begin
[V74]  273:    ClrScr; Fenster(2,2,78,23);
[9P5]  274:    WriteLn('  Berechnung der Zahl π');
[XF1]  275:    WriteLn;
[HG6]  276:    Write('  Anzahl der Nachkommastellen ');
[ØA4]  277:    Write('(bitte eingeben) = ');
[DXØ]  278:    {$I-}
[P31]  279:    repeat
[TZ1]  28Ø:      GotoXY(5Ø,3);
[N74]  281:      for i:=1 to 2Ø do Write(' ');
[A45]  282:      GotoXY(5Ø,3); Read(Stellenzahl);
[1G6]  283:    until(IOResult=Ø)AND(Ø<Stellenzahl)AND
[5A5]  284:    (Stellenzahl<=MaxStellenzahl);
[BXØ]  285:    {$I+}
[XF1]  286:    WriteLn;
[F55]  287:    Write('  Zur Fortsetzung der ');
[4D6]  288:    Write('Ausgabe Eingabe-Taste drücken!');
[XF1]  289:    WriteLn;
[AM5]  29Ø:    Blockzahl:=(Stellenzahl DIV 4)+1Ø;
[FP4]  291:    if Blockzahl>Feldlaenge then
[CT3]  292:    Blockzahl:=Feldlaenge;
[B13]  293:    if Druck=True then
[MUØ]  294:    begin
[P31]  295:      repeat
[8W1]  296:        GotoXY(1,9);
[HL5]  297:        Write('  Ist der Drucker bereit? ');
[T82]  298:        Write('(j/n)');
[9D5]  299:        ch:=ReadKey; until ch IN['j','J'];
[Y13]  3ØØ:      WriteLn; WriteLn;
[K92]  3Ø1:    end;{of if}
[N74]  3Ø2:    WriteLn('  π = '); WriteLn;
[912]  3Ø3:    Write('   3.');
[YX3]  3Ø4:    if Druck=True then begin
[ØH4]  3Ø5:      Write(lst,'Berechnung der ');
[X16]  3Ø6:      Write(lst,'Zahl π auf Stellenzahl');
[HL5]  3Ø7:      WriteLn(lst,' Nachkommastellen');
[9K5]  3Ø8:      WriteLn(lst); WriteLn(lst,'π = ');
[Ø35]  3Ø9:      WriteLn(lst); Write(lst,'  3.');
[ZNØ]  31Ø:    end;
[RX4]  311:    GetTime(Std1,Min1,Sek1,HSek1);
[GP3]  312:    Loeschen(Blockzahl,x);
[BZ4]  313:    Loeschen(Blockzahl,Ergebnis);
[AA6]  314:    ArcusTangens(16,5,Blockzahl,Ergebnis);
[445]  315:    ArcusTangens(4,239,Blockzahl,x);
[PR5]  316:    Subtraktion(Ergebnis,x,Blockzahl,
[BG6]  317:    Ergebnis);GetTime(Std2,Min2,Sek2,HSek2);
[517]  318:    Zeit; WriteLangzahl(Blockzahl-9,Ergebnis,
[GB4]  319:    Druck); WriteLn; WriteLn;
[P31]  32Ø:    repeat
[GZ3]  321:      ch:=ReadKey until ch=#13;
[GF3]  322:    Fullscreen; ClrScr;
[NE4]  323: end; {of pi_Berechnung}
[VR2]  324: procedure Menu;
[TF3]  325: var x1,y1,x2,y2: Byte;
[LX2]  326: antwort,ch: Char;
[WV1]  327: z: Boolean;
[MUØ]  328: begin
[P31]  329:    repeat
[WE5]  33Ø:      Fenster(5,6,75,2Ø); Window(7,7,7Ø,19);
[AG6]  331:      WriteLn('Wählen Sie aus:'); WriteLn;
[VK4]  332:      Write('< 1 >  Berechnung der ');
[HX4]  333:      Write('Zahl e mit Ausgabe auf ');
[PC4]  334:      WriteLn('dem Bildschirm');
[YK4]  335:      Write('< 2 >  Berechnung der ');
[RK5]  336:      Write('Zahl e mit zusätzlichem');
[HG3]  337:      WriteLn(' Ausdruck ');
[1L4]  338:      Write('< 3 >  Berechnung der ');
[D75]  339:      Write('Zahl π mit Ausgabe auf ');
[PC4]  34Ø:      WriteLn('dem Bildschirm');
[4L4]  341:      Write('< 4 >  Berechnung der ');
[LW5]  342:      Write('Zahl π mit zusätzlichem');
[A35]  343:      WriteLn(' Ausdruck '); WriteLn;
[MY4]  344:      WriteLn('< Ø >  Ende '); WriteLn;
[ES4]  345:      Write('  '); antwort:=ReadKey;
[5R2]  346:      case antwort of
```

Listing 9. »epi.pas« berechnet die Eulersche Zahl »E« und die Kreiszahl »Pi« bis auf 9999 Stellen genau

```
[UØ4]  347:        '1': e_Berechnung(False);
[UV3]  348:        '2': e_Berechnung(True);
[C44]  349:        '3': pi_Berechnung(False);
[TZ3]  35Ø:        '4': pi_Berechnung(True);
[LK2]  351:      end; {of case}
[WY2]  352:    until antwort='Ø';
[WM2]  353: end; {of Menu}
[SZ3]  354: begin {Hauptprogramm}
[L17]  355:   Fullscreen; cursoroff; Menu; Fullscreen;
[MN1]  356:   cursoron;
[KNØ]  357: end.
```

Listing 9. »epi.pas« berechnet die Eulersche Zahl »E« und die Kreiszahl »Pi« bis auf 9999 Stellen genau

Funktionstermanalyse

Turbo Pascal ist eine der leistungsfähigsten Programmiersprachen, die derzeit auf dem Markt sind. Dennoch vermißt der mathematisch interessierte Anwender immer wieder den Lösungsweg, einen Funktionsterm, der in einer Zeichenkette gespeichert ist, auszurechnen. Die Unit »fterm.pas« (Listing 10) löst dieses Problem. Sie erlaubt eine komplette Funktionstermanalyse in Turbo Pascal ab Version 4.0. Dabei werden sämtliche mathematischen Funktionen von Turbo Pascal unterstützt, aber auch einige zusätzliche. Dazu benötigt die Unit lediglich zwei Prozeduren und zwei Funktionen.

Um sich einen kurzen Eindruck von der Arbeit der Unit zu verschaffen, sollten Sie zuerst das Programm »term.exe« starten. »term.pas« (Listing 11) zeigt ihnen am Bildschirm den vollständigen Einsatz der Unit Fterm. Sie können selbst Funktionsterme eingeben. Sollten Sie dabei Fehler machen, erkennt dies das Programm. Sie erhalten eine entprechende Meldung. Mathematische Fehler, wie eine Division durch Null, werden ebenfalls berücksichtigt.

Nun zur Funktionsweise der Unit Fterm: Sie benötigen zuerst eine Variable vom Typ »String«, die den Funktionsterm speichert, außerdem noch eine Variable vom Typ »Zeiger«, die den Funktionsbaum auf dem Heap speichert. Der Typ »Zeiger« ist in der Unit deklariert. Die Unit übersetzt den Funktionsterm in einen Binärbaum, den sogenannten Funktionsbaum, der auf dem Heap abgelegt wird. Die Variable vom Typ »Zeiger« ist die Wurzel des Funktionsbaumes. Damit Sie mit der Unit arbeiten können, sollte der Programmkopf folgendermaßen aussehen:

```
program Demo;
uses fterm;
var FktTerm: string;
Wurzel: Zeiger;
```

Natürlich können Sie auch mehrere Funktionsbäume gleichzeitig bearbeiten, Sie brauchen dazu nur mehrere Variablen vom Typ »Zeiger« zu deklarieren. Im Programm sollten Sie nun zuerst einen Funktionsterm eingeben:

Fehlercode	Erklärung
0	Syntax-Fehler
1	unbekannte Funktion
2	unerwartete Klammer zu (Beispiel: SIN(x)))
3	Klammer zu fehlt (Beispiel: SIN(x)
4	Klammerebene leer (Beispiel: SIN())
5	Unerwartetes Termende (Beispiel: SIN(x) +)

Tabelle 4. Die in der Variablen »SynFehl« auftretenden Fehlercodes

Fehlercode	Erklärung
1	Division durch Null
2	Exponent einer negativen Zahl (Beispiel: (-3)^ 3)
3	Tangens von 90 Grad
4	Logarithmus einer negativen Zahl
5	Wurzel aus einer negativen Zahl

Tabelle 5. Die Fehlercodes, die aus der Variablen »RecFehl« resultieren können

```
ReadLn(FktTerm);
```

Nun muß der Funktionsterm auf syntaktische Richtigkeit überprüft werden. Dazu dient die Funktion »SynChk«. Dieser Funktion übergeben Sie die Zeichenkette mit dem Funktionsterm. Das Ergebnis ist entweder »True«, wenn der Funktionsterm richtig ist, oder »False«, wenn Sie einen Fehler gemacht haben. Sollte letzteres der Fall sein, enthält die Variable »SynFehl« einen Fehlercode. Die dabei auftretenden Fehler finden Sie in Tabelle 4.

Nachdem sichergestellt ist, daß der Funktionsterm keine Fehler enthält, muß er in einen Binärbaum umgewandelt werden. Dazu dient die Prozedur »Bbau«. Zu Beginn der Prozedur werden die Wurzel des Funktionsbaumes und der Funktionsterm übergeben. Nach Ablauf der Prozedur zeigt die Wurzel auf den Anfang des Funktionsbaumes auf dem Heap. Beachten Sie hierbei: ehe Sie die Prozedur »Bbau« aufrufen, muß die Syntax des Terms überprüft worden sein. Sind noch Fehler im Funktionsterm, stürzt der Computer wahrscheinlich ab.

Wenn der Funktionsbaum korrekt erzeugt wurde, können Sie das Ergebnis der Funktion berechnen lassen, dazu stellt die Unit Fterm die Funktion »Berech« bereit. Dieser Funktion übergeben Sie die Wurzel des Funktionsbaumes, das Ergebnis ist dann eine Zahl vom Typ Real. Hierbei ist folgendes wichtig: ehe Sie die Funktion »Berech« aufrufen, muß das Programm einen Funktionsbaum mit »Bbau« entwickelt haben. Sollten mathematische Fehler auftreten, erhalten Sie von der Funktion »Berech« ein zufälliges Ergebnis und in der Variable »RecFehl« einen Fehlercode. Die dabei eventuell entstehenden Fehlercodes finden Sie in Tabelle 5.

Der Funktionsterm kann natürlich nicht nur Zahlen als Operanden enthalten, er darf auch von einer Variablen »x« abhängen. Diese Variable »x« ist in der Unit mit dem Typ Real definiert. Wenn Sie also eine Variable benötigen, müssen Sie sie vor dem Aufruf von »Berech« definieren. Um festzustellen, ob der Funktionsterm die Variable »x« enthält, benutzen Sie einfach die Pos-Funktion. Dies ist keine Funktion der Unit Fterm, sondern eine der Standardfunktionen von Turbo Pascal.

Folgendes schematisierte Listing demonstriert noch einmal den ganzen Ablauf:

```
ReadLn(FktTerm);
```

Hiermit lesen Sie den Funktionsterm ein.

```
if SynChk(FktTerm)=True then
```

Diese Abfrage verifiziert die Richtigkeit des Funktionsterms.

```
if Pos('x',FktTerm)>0 then
```

Hiermit überprüfen Sie, ob der Funktionsterm »x« enthält, und lesen mit

```
ReadLn(x);
```

bei dem Vorhandensein von »x« mit der Anweisung

```
Bbau(Wurzel,FktTerm);
```

den Funktionsbaum ein. Mit

```
WriteLn('ergebnis: ',Berech(Wurzel));
```

berechnen Sie das Ergebnis. Die Abfrage

```
if RecFehl<>0 then WriteLn('Fehler !')
```

dient dazu, Fehler zu erkennen, wobei der Else-Zweig auch falsche Syntax abfängt:

```
else WriteLn('Syntaxfehler !');
```

Die letzte Prozedur, welche Sie für die Arbeit mit Fterm benötigen, ist die Prozedur »Bloesch«. Diese Prozedur löscht den Funktionsbaum vom Heap. Als Übergabeparameter benötigt die Prozedur nur die Wurzel des Funktionsbaums, das Löschen funktioniert also so:

```
Bloesch(Wurzel);
```

Damit ist die Funktionsweise der Unit erklärt, die Einbindung in eigene Programme dürfte kein Problem mehr sein.

Eingabe	Erklärung
+ - * /	die vier Grundrechenarten
^	berechnet die Potenz einer Zahl (x*x entspricht x+2)
SIN COS TAN	Berechnung der trigonometrischen Funktionen eines Winkels im Bogenmaß
ARCTAN	Berechnung des Arcustangens eines Winkels im Bogenmaß
RAD	berechnet das Bogenmaß eines Winkels
EXP	berechnet die Potenz der Eulerschen Zahl »E«
LN	berechnet den natürlichen Logarithmus einer Zahl
ABS	berechnet den Betrag einer Zahl
SQR	berechnet das Quadrat einer Zahl
SQRT	berechnet die Wurzel einer Zahl
FRAC	berechnet nur den Nachkommateil einer Zahl
ROUND	Rundet eine Zahl kaufmännisch auf oder ab

Tabelle 6. Die mathematischen Funktionen der Unit »Fterm«

Die interessanteste Anwendung wäre sicherlich ein Funktionsplotter, der die Kurven der eingegebenen Funktionen grafisch darstellt. Aber auch mathematische Anwendungen sind denkbar, zum Beispiel könnte ein weiteres Programm die Funktionen diskutieren. Selbst die Berechnung von Integralen wäre machbar. Dazu würde die Berechnung von Ober- und Untersummen helfen.
Nun noch ein Hinweis zur Benutzung der Unit: Die Datei »fterm.tpu« sollten Sie in Ihr Unit-Verzeichnis kopieren, damit Turbo Pascal die Unit korrekt einbinden kann. Das Unit-Verzeichnis können Sie im Menü »Options/Directories« einstellen. Tabelle 6 zeigt Ihnen abschließend in einer Übersicht die mathematischen Funktionen der Unit.

(Joachim Neisecke/et)

```
[FG3]    1: {Programm: fterm.pas
[WØ7]    2:  Funktion: Unit berechnet Funktionsterme
[8Z5]    3:  Sprache:  Turbo Pascal ab Version 5.5
[V44]    4:  Autor:    Joachim Neisecke
[TX4]    5:  (c)1991 DMV Widuch GmbH & Co. KG}
[A12]    6: unit Fterm;
[H93]    7: interface uses Crt;
[GT3]    8: type Zeiger=^Element;
[EG2]    9:      Element=record
[2K4]   1Ø:        Operator: Char; Zahl: real;
[7V2]   11:        lks,rts: Zeiger
[ZNØ]   12:      end;
[NV5]   13: var x: real; SynFehl,RecFehl: Byte;
[ZG7]   14: function SynChk(var term: string): Boolean;
[8B5]   15: function Berech(p: Zeiger): real;
[ØD7]   16: procedure Bbau(var wurzel:Zeiger;f:string);
[TK5]   17: procedure Bloesch(var p: Zeiger);
[4L2]   18: implementation
[G66]   19: const OfnKlam: Set of Char=['(','[','{'];
[XH5]   2Ø:       ZuKlam: Set of Char=[')',']','}'];
[296]   21: function holTeil(term: string): Byte;
[YD4]   22: var Klamebn: Byte; I: Byte;
[MUØ]   23: begin
[1M2]   24:   Klamebn:=1; I:=1;
[P31]   25:   repeat
[V86]   26:     if term[I]IN OfnKlam then Inc(Klamebn)
[G94]   27:     else if term[I]IN ZuKlam
[9X2]   28:          then Dec(Klamebn);
[951]   29:     Inc(I);
[2Y4]   3Ø:   until Klamebn=Ø; holTeil:=I-2;
[ZNØ]   31: end;
[H87]   32: procedure Vrzchn_entf(var formel: string);
[ZH2]   33: var I: Integer;
[MUØ]   34: begin
[P31]   35:   repeat
[WY2]   36:     I:=Pos('(-',formel);
[4U4]   37:     if I=Ø then I:=Pos('{-',formel);
[4S4]   38:     if I=Ø then I:=Pos('[-',formel);
[4A5]   39:     if I>Ø then Insert('Ø',formel,I+1);
[RM1]   4Ø:   until I=Ø;
[P31]   41:   repeat
[PY2]   42:     I:=Pos('(+',formel);
[YT4]   43:     if I=Ø then I:=Pos('{+',formel);
[YR4]   44:     if I=Ø then I:=Pos('[+',formel);
[4A5]   45:     if I>Ø then Insert('Ø',formel,I+1);
[RM1]   46:   until I=Ø;
[U56]   47:   if(formel[1]='+')OR(formel[1]='-')then
[B63]   48:      Insert('Ø',formel,1);
[ZNØ]   49: end;
[596]   5Ø: function hl_Tkn(funkt: string): Char;
[MUØ]   51: begin
[EP5]   52:   if funkt='ABS' then hl_Tkn:='A' else
[9R5]   53:   if funkt='COS' then hl_Tkn:='C' else
[9U5]   54:   if funkt='SIN' then hl_Tkn:='S' else
[8U5]   55:   if funkt='TAN' then hl_Tkn:='T' else
[WR5]   56:   if funkt='RAD' then hl_Tkn:='D' else
[UR5]   57:   if funkt='EXP' then hl_Tkn:='E' else
[TK5]   58:   if funkt='LN' then hl_Tkn:='L' else
[LU5]   59:   if funkt='SQR' then hl_Tkn:='Q' else
[8P6]   6Ø:   if funkt='SQRT' then hl_Tkn:='/' else
[CV5]   61:   if funkt='FRAC' then hl_Tkn:='F' else
[836]   62:   if funkt='ROUND' then hl_Tkn:='R' else
[986]   63:   if funkt='ARCTAN' then hl_Tkn:='N' else
[ZN1]   64:   hl_Tkn:=#Ø
[ZNØ]   65: end;
[ZG7]   66: function SynChk(var term: string): Boolean;
[543]   67: var Zstnd,I: Byte;
[PH6]   68: EG,Laenge: Byte; Zeichen: Char; fe: Byte;
[FL5]   69: Klammerstack: array[1..2Ø]of Char;
[WP2]   7Ø: Klammersp: Byte;
[PG2]   71: const ok=7; f=8;
[9M1]   72:       zstndtab:
[ØR3]   73:       array[Ø..6,Ø..7]of Byte=
[VE2]   74: ((1,f,f,2,f,6,4,f),
[N52]   75: (1,f,f,2,f,6,4,f),
[2D2]   76: (f,5,Ø,2,3,f,f,ok),
[U92]   77: (f,f,f,2,f,f,f,f),
[FF2]   78: (f,5,Ø,f,f,f,f,ok),
[FF2]   79: (f,5,Ø,f,f,f,f,ok),
[KF2]   8Ø: (1,f,f,f,f,f,f,f));
[MUØ]   81: begin
[HZ1]   82:   SynFehl:=Ø;
[X33]   83:   while Pos(' ',term)>Ø
[FR4]   84:   do Delete(term,Pos(' ',term),1);
[EG4]   85:   for I:=1 to Ord(term[Ø])do
[XN3]   86:   if(term[I]IN['a'..'z'])
[SN3]   87:   AND(term[I]<>'x')then
[R95]   88:   term[I]:=UpCase(term[I])else
[494]   89:   if(term[I]='x')AND(I>1)then
[873]   9Ø:   if term[I-1]='E' then
[EN4]   91:   term[I]:=UpCase(term[I]);
[FB3]   92:   Vrzchn_entf(term);
[8E2]   93:   I:=1; Zstnd:=Ø;
[CF3]   94:   Laenge:=Length(term);
[R52]   95:   Klammersp:=Ø;
[P31]   96:   repeat
[T13]   97:     Zeichen:=term[I];
[951]   98:     Inc(I);
[654]   99:     if Zeichen IN OfnKlam then
[MUØ]  1ØØ:     begin
[6M2]  1Ø1:       Inc(Klammersp);
[ZY5]  1Ø2:       Klammerstack[Klammersp]:=Zeichen;
[ZNØ]  1Ø3:     end;
[NZ3]  1Ø4:     if Zeichen IN ZuKlam then
[Y43]  1Ø5:     if Klammersp>Ø then
[MUØ]  1Ø6:     begin
[9Z2]  1Ø7:       Delete(term,I-1,1);
[7F5]  1Ø8:       case Klammerstack[Klammersp]of
[KR3]  1Ø9:         '(': Insert(')',term,I-1);
[714]  11Ø:         '[': Insert(']',term,I-1);
[P14]  111:         '{': Insert('}',term,I-1)
[ZNØ]  112:       end;
[NB2]  113:       Dec(Klammersp)
[ØC2]  114:     end else begin
[KE3]  115:       SynFehl:=2; Zstnd:=f
[ZNØ]  116:     end;
[NK4]  117:   until(I>Laenge)OR(Zstnd=f);
[KØ4]  118:   if Klammersp>Ø then begin
[NE3]  119:     SynFehl:=3; Zstnd:=f
[ZNØ]  12Ø:   end;
[VPØ]  121:   I:=1;
[116]  122:   while(I<=Laenge)AND(Zstnd<f)do begin
[T13]  123:     Zeichen:=term[I];
[AM2]  124:     case Zeichen of
[2E2]  125:       '(','{','[':EG:=Ø;
[RN2]  126:       ')','}',']': EG:=1;
[A83]  127:       '+','-','*','/','^': EG:=2;
[5Z1]  128:       'Ø'..'9': EG:=3;
[NG1]  129:       '.': EG:=4;
[TM1]  13Ø:       'x': EG:=6;
[ML2]  131:       else begin fe:=Ø;
[VA3]  132:         while(fe<=Laenge)AND
[5K5]  133:         (term[I+fe]IN['A'..'Z'])do Inc(fe);
[RF4]  134:         if fe+I>Laenge then Zstnd:=f
[WP1]  135:         else begin
[FK5]  136:           Zeichen:=hl_Tkn(Copy(term,I,fe));
[NG4]  137:           if Zeichen=#Ø then SynFehl:=1
[WP1]  138:           else begin
[SC2]  139:             EG:=5; I:=I+fe-1;
[4MØ]  14Ø:           end
[4MØ]  141:         end
[ZNØ]  142:       end;
[ZNØ]  143:     end;
[XN5]  144:     if(Zstnd=6)AND(EG>Ø)then SynFehl:=2;
[RN5]  145:     if(Zstnd=1)AND(EG=1)then SynFehl:=4;
[SH4]  146:     if SynFehl>Ø then Zstnd:=f;
[8N2]  147:     if Zstnd<>f then
[SP4]  148:       Zstnd:=zstndtab[Zstnd,EG];
[951]  149:     Inc(I);
[ZNØ]  15Ø:   end;
[8N2]  151:   if Zstnd<>f then
```

Listing 10. »fterm.pas«: Die große Unit, die Funktionsterme berechnet

```
[MUØ]  152:    begin
[1E4]  153:      Zstnd:=zstndtab[Zstnd,7];
[7A4]  154:      if Zstnd=f then SynFehl:=5
[ZNØ]  155:    end;
[8N2]  156:    if Zstnd<>f then
[LG5]  157:       SynChk:=True else SynChk:=false
[ZNØ]  158: end;
[9G4]  159: function Elementlieferung
[ØF4]  16Ø:          (var term: string): Zeiger;
[U82]  161: var p: Zeiger;
[PX2]  162: strzahl: string;
[GM2]  163: dummy: Integer;
[5Ø3]  164: klammerpos: Byte;
[GN3]  165: const rechen_zeichen:
[TN4]  166:       Set of Char=['+','-','*','/','^'];
[Z85]  167:       ziffer: Set of Char=['Ø'..'9','.'];
[MUØ]  168: begin
[243]  169:   strzahl:=''; New(p);
[UC4]  17Ø:   p^.lks:=NIL; p^.rts:=NIL;
[WB4]  171:   if Length(term)=Ø then begin
[MT4]  172:     p^.Operator:=#255; p^.Zahl:=Ø;
[3D1]  173:   end else
[LX4]  174:   begin if term[1]IN ziffer then
[MUØ]  175:     begin
[L54]  176:       while(term[1]IN ziffer)
[VD4]  177:       AND(Length(term)>Ø)do begin
[UF4]  178:         strzahl:=strzahl+term[1];
[ØL2]  179:         Delete(term,1,1);
[ZNØ]  18Ø:       end;
[4V4]  181:       Val(strzahl,p^.Zahl,dummy);
[LL2]  182:       p^.Operator:=#Ø
[3D1]  183:     end else
[SH5]  184:     if term[1]IN rechen_zeichen then
[MT4]  185:     begin p^.Operator:=term[1];
[GC4]  186:       p^.Zahl:=Ø; Delete(term,1,1)
[3D1]  187:     end else
[D64]  188:     if term[1]IN OfnKlam then
[4U3]  189:     begin p^.Operator:='(';
[GC4]  19Ø:       p^.Zahl:=Ø; Delete(term,1,1)
[3D1]  191:     end else
[ZY3]  192:     if term[1]='x' then begin
[GU4]  193:       p^.Operator:='X'; p^.Zahl:=Ø;
[ØL2]  194:       Delete(term,1,1);
[ØC2]  195:     end else begin
[VC2]  196:       klammerpos:=1;
[8H6]  197:       while NOT(term[klammerpos]IN OfnKlam)
[643]  198:             do Inc(klammerpos);
[LR4]  199:       {offene Klammern suchen !}
[KL3]  2ØØ:       p^.Operator:=hl_Tkn
[3H4]  2Ø1:             (Copy(term,1,klammerpos-1));
[6W1]  2Ø2:       p^.Zahl:=Ø;
[BG4]  2Ø3:       Delete(term,1,klammerpos-1)
[4MØ]  2Ø4:     end
[EE4]  2Ø5:   end;{of if length(term)=Ø}
[5L3]  2Ø6:   Elementlieferung:=p;
[ZNØ]  2Ø7: end;
[295]  2Ø8: function bwert(p: Zeiger): real;
[2Ø5]  2Ø9: var Basis,Divisor,Wert: real;
[MUØ]  21Ø: begin
[726]  211:   if p^.Operator=#Ø then bwert:=p^.Zahl
[N96]  212:   else if p^.Operator='X' then bwert:=x
[124]  213:   else case p^.Operator of
[BF6]  214:    '+': bwert:=bwert(p^.lks)+bwert(p^.rts);
[RF6]  215:    '-': bwert:=bwert(p^.lks)-bwert(p^.rts);
[5F6]  216:    '*': bwert:=bwert(p^.lks)*bwert(p^.rts);
[565]  217:    'S': bwert:=Sin(bwert(p^.rts));
[V45]  218:    'C': bwert:=Cos(bwert(p^.rts));
[E65]  219:    'D': bwert:=bwert(p^.rts)*Pi/18Ø;
[M55]  22Ø:    'E': bwert:=Exp(bwert(p^.rts));
[A35]  221:    'A': bwert:=Abs(bwert(p^.rts));
[775]  222:    'Q': bwert:=Sqr(bwert(p^.rts));
[M25]  223:    'F': bwert:=Frac(bwert(p^.rts));
[SD5]  224:    'R': bwert:=Round(bwert(p^.rts));
[RN5]  225:    'N': bwert:=ArcTan(bwert(p^.rts));
[HDØ]  226:    '/':
[125]  227:    begin Divisor:=bwert(p^.rts);
[PV3]  228:      if Divisor=Ø then begin
[GP3]  229:        RecFehl:=1; Divisor:=1
[ZNØ]  23Ø:      end;
[515]  231:      bwert:=bwert(p^.lks)/Divisor
[ZNØ]  232:    end;
[YHØ]  233:    '^':
[MUØ]  234:     begin
[9P3]  235:       Basis:=bwert(p^.lks);
[Z73]  236:       if Basis<Ø then begin
[283]  237:         RecFehl:=2; Basis:=1
[ZNØ]  238:       end;
[D26]  239:       bwert:=Exp(bwert(p^.rts)*Ln(Basis))
[ZNØ]  24Ø:     end;
[ØHØ]  241:     'T':
[WA5]  242:     begin Wert:=Cos(bwert(p^.rts));
[D15]  243:       if Wert=Ø then begin RecFehl:=3;
[881]  244:         Wert:=1
[ZNØ]  245:       end;
[Y85]  246:       bwert:=Sin(bwert(p^.rts))/Wert
[ZNØ]  247:     end;
[8GØ]  248:     'L':
[ZD4]  249:     begin Wert:=bwert(p^.rts);
[G93]  25Ø:       if Wert<=Ø then begin
[Z53]  251:         RecFehl:=4; Wert:=2
[ZNØ]  252:       end;
```

```
[YL2]  253:       bwert:=Ln(Wert)
[ZNØ]  254:     end;
[NØ1]  255:     '/':
[ZD4]  256:     begin Wert:=bwert(p^.rts);
[653]  257:       if Wert<Ø then begin
[Y53]  258:         RecFehl:=5; Wert:=1
[ZNØ]  259:       end;
[7Ø3]  26Ø:       bwert:=Sqrt(Wert)
[4MØ]  261:     end
[4MØ]  262:   end
[ZNØ]  263: end;
[8B5]  264: function Berech(p: Zeiger): real;
[6F5]  265: begin RecFehl:=Ø; Berech:=bwert(p)
[ZNØ]  266: end;
[W77]  267: function Wertigkeit(Operator: Char): Byte;
[AN3]  268: begin case Operator of
[ZD3]  269:     '+','-': Wertigkeit:=1;
[5E3]  27Ø:     '*','/': Wertigkeit:=2;
[163]  271:     '^': Wertigkeit:=3;
[513]  272:     '(': Wertigkeit:=5;
[T43]  273:     else Wertigkeit:=4
[4MØ]  274:   end
[ZNØ]  275: end;
[ØD7]  276: procedure Bbau(var wurzel:Zeiger;f:string);
[4S2]  277: var p,q: Zeiger;
[KØ7]  278: altWert,neuWert: Byte; operand: Boolean;
[SV2]  279: ZgOprnd: Zeiger;
[ND5]  28Ø: WertStuf: array[1..3]of Zeiger;
[YL5]  281: const END_OF_TERM=#255; Klamebn=5;
[ET5]  282:       Funktion=4; Exponent=3; ADD_SUB=1;
[SZ4]  283:       NoRecZchn: Set of Char=[#Ø,'X'];
[SF7]  284: function Zeiger_auf_letztes_Blatt: Zeiger;
[W82]  285: var q: Zeiger;
[MUØ]  286: begin
[HT1]  287:   q:=wurzel;
[C95]  288:   while q^.rts<>NIL do q:=q^.rts;
[ØB5]  289:   Zeiger_auf_letztes_Blatt:=q
[ZNØ]  29Ø: end;
[MUØ]  291: begin
[P43]  292:   WertStuf[1]:=NIL;
[T43]  293:   WertStuf[2]:=NIL;
[W43]  294:   WertStuf[3]:=NIL;
[R92]  295:   ZgOprnd:=NIL;
[DØ2]  296:   altWert:=Ø;
[7N2]  297:   operand:=false;
[2X4]  298:   wurzel:=Elementlieferung(f);
[D17]  299:   if NOT(wurzel^.Operator IN NoRecZchn)then
[MUØ]  3ØØ:   begin
[SF2]  3Ø1:     operand:=True;
[RN4]  3Ø2:     if wurzel^.Operator='(' then
[MUØ]  3Ø3:     begin
[G45]  3Ø4:       Dispose(wurzel); wurzel:=NIL;
[XG5]  3Ø5:       Bbau(wurzel,Copy(f,1,holTeil(f)));
[NN3]  3Ø6:       Delete(f,1,holTeil(f)+1);
[ZNØ]  3Ø7:     end;
[ØØ3]  3Ø8:     ZgOprnd:=wurzel;
[ZNØ]  3Ø9:   end;
[P31]  31Ø:   repeat
[5Ø4]  311:     p:=Elementlieferung(f);
[TL5]  312:     if p^.Operator<>END_OF_TERM then
[MUØ]  313:     begin
[F95]  314:       if p^.Operator IN NoRecZchn then
[MUØ]  315:       begin
[785]  316:         q:=Zeiger_auf_letztes_Blatt;
[CH1]  317:         q^.rts:=p
[3D1]  318:       end else
[MUØ]  319:       begin
[1Ø6]  32Ø:         neuWert:=Wertigkeit(p^.Operator);
[F14]  321:         if neuWert=ADD_SUB then
[TN3]  322:         if WertStuf[ADD_SUB]=
[M93]  323:            NIL then altWert:=Ø
[DW3]  324:         else altWert:=ADD_SUB;
[994]  325:         if neuWert>Exponent then
[W33]  326:            operand:=True else
[474]  327:         if(neuWert<Exponent)AND
[W24]  328:           (operand=True)then begin
[8W4]  329:           operand:=false; ZgOprnd:=NIL
[ZNØ]  33Ø:         end;
[HB4]  331:         if operand=True then begin
[3R5]  332:           case neuWert of Funktion,Klamebn:
[MUØ]  333:             begin
[WU4]  334:               if neuWert=Klamebn then begin
[B73]  335:                 Dispose(p); p:=NIL;
[3NØ]  336:                 Bbau
[NU3]  337:                 (p,Copy(f,1,holTeil(f)));
[ZH3]  338:                 Delete(f,1,holTeil(f)+1)
[ZNØ]  339:               end;
[785]  34Ø:               q:=Zeiger_auf_letztes_Blatt;
[XP1]  341:               q^.rts:=p;
[BB2]  342:               if ZgOprnd=NIL
[SS2]  343:                 then ZgOprnd:=p
[ZNØ]  344:             end;
[CM1]  345:             Exponent:
[FU3]  346:             if ZgOprnd=wurzel then
[8U3]  347:             begin p^.lks:=ZgOprnd;
[7K3]  348:               wurzel:=p; ZgOprnd:=p
[ØC2]  349:             end else begin
[HT1]  35Ø:               q:=wurzel;
[364]  351:               while q^.rts<>ZgOprnd do
[H94]  352:               q:=q^.rts; p^.lks:=q^.rts;
[HF3]  353:               q^.rts:=p; ZgOprnd:=p
```

```
[4MØ]   354:              end
[4MØ]   355:            end
[ØC2]   356:          end else begin
[9X3]   357:            if altWert=Ø then begin
[L33]   358:              altWert:=neuWert;
[EL4]   359:              pˆ.lks:=wurzel; wurzel:=p;
[FS4]   36Ø:              WertStuf[neuWert]:=wurzel
[3D1]   361:            end else
[575]   362:            if neuWert<=altWert then begin
[A45]   363:              if WertStuf[altWert]=wurzel
[ZN1]   364:              then begin
[8U4]   365:                pˆ.lks:=WertStuf[altWert];
[LT1]   366:                wurzel:=p;
[ØY4]   367:                WertStuf[neuWert]:=wurzel;
[ØC2]   368:              end else begin
[HT1]   369:                q:=wurzel;
[9D2]   37Ø:                while qˆ.rts<>
[593]   371:                WertStuf[altWert]
[LV4]   372:                do q:=qˆ.rts; pˆ.lks:=qˆ.rts;
[XP1]   373:                qˆ.rts:=p;
[3T3]   374:                WertStuf[neuWert]:=p
[ZNØ]   375:              end;
[XZ2]   376:              altWert:=neuWert
[ØC2]   377:            end else begin
[8U5]   378:              pˆ.lks:=WertStuf[altWert]ˆ.rts;
[ØV4]   379:              WertStuf[altWert]ˆ.rts:=p;
[TW3]   38Ø:              WertStuf[neuWert]:=p;
[XZ2]   381:              altWert:=neuWert
[4MØ]   382:            end
[4MØ]   383:          end
[4MØ]   384:        end
[ZNØ]   385:      end;
[6F5]   386:    until pˆ.Operator=END_OF_TERM;
[TY1]   387:    Dispose(p);
[ZNØ]   388: end;
[TK5]   389: procedure Bloesch(var p: Zeiger);
[MUØ]   39Ø: begin
[HS4]   391:   if(pˆ.lks=NIL)AND(pˆ.rts=NIL)
[3B4]   392:      then Dispose(p) else begin
[VW5]   393:     if pˆ.rts<>NIL then Bloesch(pˆ.rts);
[7T5]   394:     if pˆ.lks<>NIL then Bloesch(pˆ.lks)
[ZNØ]   395:   end;
[K71]   396:   p:=NIL;
[ZNØ]   397: end;
[MUØ]   398: begin
[UH4]   399:   SynFehl:=Ø; RecFehl:=Ø; x:=Ø
[KNØ]   4ØØ: end.
```

Listing 10. »fterm.pas«: Die große Unit, die Funktionsterme berechnet

```
[ZF3]     1: {Programm: term.pas
[ZZ4]     2:  Funktion: Demo für Unit Fterm
[BV5]     3:  Sprache: Turbo Pascal ab Version 4.Ø
[9X3]     4:  Autor: Joachim Neisecke
[TX4]     5:  (c)1991 DMV Widuch GmbH & Co. KG}
[RB5]     6: program Funktionstermanalyse;
[5R2]     7: uses Crt,fterm;
[4M3]     8: var FktTerm: string;
[R34]     9:     ch: Char; wurzel: zeiger;
[HZ6]    1Ø: const S_Fehler: array[Ø..5]of string[25]=
[DL2]    11:       ('Syntaxfehler',
[XT3]    12:       'unbekannte Funktion',
[FB4]    13:       'unerwartete Klammer zu',
[X63]    14:       'Klammer zu fehlt',
[ØK2]    15:       'Klammern leer',
[F74]    16:       'unerwartetes Termende');
[M56]    17:       R_Fehler: array[1..5]of string[33]=
[1S3]    18:       ('Division durch Null',
[Z45]    19:       'Potenz einer negativen Zahl',
[5Y2]    2Ø:       'Tangens von 9Ø°',
[PW5]    21:       'Logarithmus einer negativen Zahl',
[4C5]    22:       'Wurzel einer negativen Zahl');
[MUØ]    23:       begin
[DB1]    24:         ClrScr;
[WZ1]    25:         GotoXY(27,1);
[F65]    26:         Write('Funktionstermanalyse');
[ØØ2]    27:         GotoXY(27,2);
[EE4]    28:         Write('====================');
[TV1]    29:         GotoXY(1,4);
[7E5]    3Ø:         WriteLn('     Dieses Programm ber',
[3W5]    31:         'echnet einen beliebigen Funktion',
[A73]    32:         'sterm. Erlaubte');
[SM5]    33:         WriteLn('     Rechenoperationen s',
[AZ4]    34:         'ind rechts unten angegeben.',
[LP3]    35:         ' Klammern werden vom');
[HZ5]    36:         WriteLn('     Programm unterstütz',
[1M5]    37:         't. Die Einbindung einer Variable',
[LH1]    38:         ' x ist');
[ES5]    39:         WriteLn('     ebenfalls möglich.');
[7T2]    4Ø:         Window(61,9,8Ø,25);
[W14]    41:         WriteLn('Funktionen :');
[KN3]    42:         WriteLn('============');
[5B4]    43:         WriteLn;WriteLn('+ - * / ^');
[WF4]    44:         WriteLn('SIN COS TAN RAD');
[G33]    45:         WriteLn('LN EXP');
[KC3]    46:         WriteLn('SQR SQRT');
[YP3]    47:         WriteLn('FRAC ROUND');
[YL5]    48:         WriteLn('ABS'); WriteLn('ARCTAN');
```

```
[XF1]    49:         WriteLn;
[ZL3]    5Ø:         WriteLn('Klammern :');
[A15]    51:         WriteLn('==========='); WriteLn;
[SX2]    52:         Write('( [ { } ] )');
[6K2]    53:         Window(1,9,6Ø,25);
[P31]    54:         repeat
[P31]    55:           repeat
[6G3]    56:             FktTerm:=''; WriteLn;
[T15]    57:             WriteLn('Bitte geben Sie nun ',
[XM4]    58:             'einen Funktionsterm ein :');
[BV4]    59:             Write('f(x)=');ReadLn(FktTerm);
[TL4]    6Ø:             Write('Funktionsterm ist ');
[PX3]    61:             if SynChk(FktTerm)=True
[EM4]    62:                then WriteLn('gültig !')else
[MUØ]    63:             begin
[EX3]    64:               WriteLn('ungültig ! ');
[BA4]    65:               WriteLn('Fehlernummer : ',
[DF1]    66:               SynFehl,
[MB4]    67:               ' (',S_Fehler[SynFehl],')')
[ZNØ]    68:             end;
[SU4]    69:           until SynChk(FktTerm)=True;
[1R3]    7Ø:           Bbau(wurzel,FktTerm);
[P31]    71:           repeat
[XF1]    72:             WriteLn;
[DØ4]    73:             if Pos('x',FktTerm)>Ø then
[MUØ]    74:             begin
[6G4]    75:               Write('Bitte geben Sie eine',
[WH3]    76:               'n Wert für x ein :');
[H43]    77:               ReadLn(x); WriteLn;
[ZNØ]    78:             end;
[TL3]    79:             Write('f(x)=',FktTerm);
[DØ4]    8Ø:             if Pos('x',FktTerm)>Ø then
[MZ3]    81:                WriteLn(' und x=',x: 6: 2)
[ØA2]    82:             else WriteLn;
[M95]    83:             WriteLn('Der Funktionswert f(',
[YF1]    84:             'x) ist :'
[9E3]    85:             ,Berech(wurzel):1Ø:2);
[ZN2]    86:             if RecFehl>Ø then
[R84]    87:             WriteLn('Rechenfehler : ',
[3C1]    88:             RecFehl,
[YB4]    89:             ' (',R_Fehler[RecFehl],')');
[GW4]    9Ø:             Write('Neue Berechnung (j/n) ',
[UL2]    91:             '?');ch:=ReadKey;
[HH3]    92:             WriteLn; until ch='n';
[WT2]    93:           BLoesch(wurzel);
[695]    94:           Write('Neue Funktion eingeben (',
[NW4]    95:           'j/n) ?');ch:=ReadKey; WriteLn;
[X12]    96:         until ch='n';
[KNØ]    97:       end.
```

Listing 11. »term.pas« führt die Arbeit von Fterm vor

Anzeige des ANSI-Treibers

Um einen Weg zur einfachen, von einer bestimmten Grafikkarte unabhängigen, Bildschirmmanipulation zu schaffen, haben die Systemprogrammierer von MS-DOS den Treiber »ansi.sys« geschaffen. Erst dieser Treiber gestattet die Ausnutzung von farbiger Bildschirmausgabe und von Tastaturmakros in Stapeldateien.

Wenn ANSI-Befehle ausgeführt werden, ohne daß Sie »ansi.sys« geladen haben, entsteht ein für den Benutzer unlesbarer »Zeichensalat« auf dem Bildschirm. Wenn Sie also ANSI-Befehle in Stapeldateien und Programmen verwenden, sollten Sie testen, ob Sie den ANSI-Treiber geladen haben. Die Turbo-Pascal-Unit »ansi.pas« (Listing 12) enthält die Funktion »If_Ansi_Install«, welche »True« ist, wenn »ansi.sys« geladen wurde. Ihre Arbeitsweise führt Ihnen »ansitest.pas« (Listing 13) vor.

Um zu ermitteln, ob »ansi.sys« als Treiber von MS-DOS eingetragen ist, müssen Sie zunächst über die interne, bisher nicht offiziell dokumentierte Funktion 52hex des Interrupts 21hex einen Zeiger auf den internen Datenblock von MS-DOS ermitteln. Die Register »ES:BX« dienen als Zeiger. Tabelle 7 zeigt, wie der Datenblock aufgebaut ist.

Für die weitere Betrachtung ist nur der Zeiger auf den ANSI-Treiber wichtig. In der Liste taucht er als Zeiger auf den Konsolentreiber auf. Über diesen Treiber wickelt MS-DOS sämtliche Ein- und Ausgaben über Tastatur und Bildschirm ab. Nicht wenige Programme umgehen allerdings die doch ziemlich langsamen Funktionen von MS-DOS, so daß nur »sauber« programmierte Software den ANSI-Treiber nutzen kann. Ist die Offsetadresse des Zeigers gleich Null, so ist »ansi.sys« geladen. Der interne Treiber von MS-DOS besitzt eine Offsetadresse ungleich Null. Da diese Me-

Adresse	Position	Inhalt
ES:	(BX-4)	Zeiger auf MCB (Memory Control Block)
ES:	BX	Zeiger auf DPB (DOS Parameter Block)
ES:	(BX+4)	Zeiger auf Beginn des internen Puffers
ES:	(BX+8)	Zeiger auf Clock Treiber
ES:	(BX+12)	Zeiger auf Consol Treiber
ES:	(BX+16)	Word Maximale Sektorlänge in Bytes
ES:	(BX+18)	Zeiger auf Ende des internen Puffers
ES:	(BX+22)	Zeiger auf Umleitungstabelle
ES:	(BX+26)	Zeiger auf Dateitabelle

Tabelle 7. Die Register »ES:BX« dienen als Zeiger

thode nicht nach dem Namen »ansi.sys« sucht, arbeitet sie auch mit den verbesserten Bildschirmtreibern, welche Sie zu modernen EGA- und VGA-Karten erhalten, zusammen.

Wenn Sie testen wollen, ob ein Programm auch ohne »ansi.sys« arbeitet, ohne die Datei »config.sys« verändern zu müssen, können Sie die Prozeduren »Ansi_Off« und »Ansi_On« verwenden. Diese Prozeduren schalten »ansi.sys« ein und aus.

Für die interne Bearbeitung der ANSI-Befehle ist der Interrupt 29hex zuständig. Wollen Sie Ansi nun ausschalten, brauchen Sie nur den Interrupt 29hex so verändern, daß dieser alle Aufrufe auf Interrupt 10hex umleitet. Für das spätere Einschalten wird der alte Zeiger auf Interrupt 29hex gespeichert. Der neue Interrupt 29hex sowie der Zeiger auf den alten Interrupt 29hex werden im Speicher im Bereich von 0:4F0hex bis 0:4FFhex gespeichert, damit das Aus- und Einschalten von »ansi.sys« nicht unbedingt in einem Programm erfolgen muß.

(Michael Humpl/et)

```
[HG3]    1: { Programm: ansi.pas
[HK5]    2:  Sprache:  Turbo Pascal Version 5.5
[XR3]    3:  Autor:    Michael Humpl
[6V4]    4:  (c)1991 DMV Widuch GmbH & Co.KG)
[XP1]    5: unit ANSI;
[SK1]    6: interface
[ØØ6]    7: function If_Ansi_Install: Boolean;
[3F3]    8: procedure Ansi_Off;
[673]    9: procedure Ansi_On;
[4L2]   1Ø: Implementation
[S62]   11: uses Dos,Crt;
[ØØ6]   12: function If_Ansi_Install: Boolean;
[TU2]   13: var r: Registers;
[MUØ]   14: begin
[ZB1]   15:   r.AH:=$52;
[ØS1]   16:   Intr($21,r);
[Z43]   17:   If_Ansi_Install:=
[XM3]   18: (Mem[r.ES: r.BX+$C]=Ø);
[ZNØ]   19: end;
[3F3]   2Ø: procedure Ansi_Off;
[NN5]   21: var save: pointer absolute $Ø: $4FØ;
[3K5]   22: const newdrv: array[1..7]of Byte=
[YH3]   23: ($5Ø,$B4,$ØE,$CD,$1Ø,$58,$CF);
[MUØ]   24: begin
[863]   25:   GetIntVec($29,save);
[FD4]   26:   Move(Mem[$Ø: $4F2],newdrv,7);
[VK4]   27:   SetIntVec($29,@Mem[$Ø: $4F2]);
[ZNØ]   28: end;
[673]   29: procedure Ansi_On;
[NN5]   3Ø: var save: pointer absolute $Ø: $4FØ;
[MUØ]   31: begin
[Ø73]   32:   SetIntVec($29,save);
[ZNØ]   33: end;
[KNØ]   34: end.
```

Listing 12. Die Unit »ansi.pas« gibt Auskunft darüber, ob der ANSI-Treiber geladen ist

```
[233]    1: Program ansitest;
[VP1]    2: uses ansi;
[MUØ]    3: begin
[XF1]    4:   WriteLn;
[UG6]    5:   Write('Der Einheitentreiber ANSI ist ');
[7T4]    6:   if NOT(If_ansi_install)then
[BB2]    7:   Write('nicht ');
[SØ4]    8:   WriteLn('installiert!');
[XF1]    9:   WriteLn;
[KNØ]   1Ø: end.
```

Listing 13. »ansitest.pas« führt die Arbeit der Unit »ansi.pas« vor

Kopierschutz im Diskettenformat

Wer schon einmal ein Programm geschrieben hat und es vor dem Kopieren schützen wollte, hat gemerkt, daß dieser Vorgang schwierig und kompliziert ist. Doch mit dem Gebrauch der Programme D_schutz, Disktest und Testorg erleichtern Sie sich diese Arbeit.

»d_schutz.pas« (Listing 14) formatiert die Spuren 38 und 39, jeweils Seite 0, einer 360-KByte-Diskette auf ein Format, das Diskcopy nicht lesen kann. Auch professionelle Programme wie Star Manager und PC-Tools können dieses Format nicht lesen. Diese Formatierung ereichen Sie dadurch, daß Sie die Disk-Parametertabelle von MS-DOS verändern. Der Interrupt 1Ehex ist der Zeiger, der auf Tabelle 8 verweist.

Wenn Sie nun die Offsets 3 und 4 verändern, können Sie eine Formatierung erreichen, die nur dann geschrieben und gelesen werden kann, wenn in der Diskettenparametertabelle die gleichen Werte enthalten sind. Da die meisten Diskettenkopierprogramme die von MS-DOS vorgegebene Tabelle benutzen, kann kaum jemand die anderweitig formatierten Spuren lesen.

Versucht also jemand, eine durch D_schutz geschützte Diskette mit Diskcopy zu kopieren, so kann Diskcopy die Spuren 38 und 39, Seite 0, nicht lesen. Die entsprechenden Spuren werden auf der Zieldiskette »normal« formatiert.

»disktest.pas« (Listing 15) verwendet die Unit »test_org.pas« (Listing 16), um zu testen, ob die von D_Schutz formatierten Spuren noch vorhanden sind. Die Unit enthält eine Prozedur, welche die Parametertabelle wieder umstellt und versucht, die Spuren 38 und 39 zu lesen.

Die Prozedur »Test« übernehmen Sie in Ihre eigenen Programme. Ab diesem Zeitpunkt läuft das Programm nur noch auf einer Diskette, die Sie mit D_Schutz geschützt haben. Es gibt keine Kopie dieser Diskette, die noch läuft.

(Michael Humpl/et)

Offset	Inhalt	Bedeutung
0	223	Wartezeit zwischen Steps beim Schrittmotor
1	2	Wartezeit zwischen Datenzugriffen
2	37	Motornachlaufzeit
3	2	Bytes je Sektor in 256 Byteblöcken
4	9	Sektoren je Spur
5	42	Raum zwischen Sektor ID und Daten
6	255	Datenblocklänge zur Übertragung an Diskette
7	80	Raum zwischen Sektoren beim Formatieren
8	246	Füllbyte für die Formatierung
9	15	Wartezeit zwischen »Kopf über Sektor« und »Sektor Lesen« in Millisekunden
10	2	Wartezeit zwischen »Motor ein« und »Daten lesen« in 1/8 Sekunden

Tabelle 8. Der Interrupt 1Ehex dient als Zeiger auf diese Tabelle

```
[MA4]    1: {Programm: d_schutz.pas
[5Z3]    2:  Funktion: Kopierschutz
[LV5]    3:  Sprache: Turbo Pascal ab Version 5.5
[GM3]    4:  Autor:   Michael Humpl
[TX4]    5:  (c)1991 DMV Widuch GmbH & Co. KG)
[S62]    6: uses Crt,Dos;
[3U3]    7: const format_tabelle1:
[SF3]    8:       array[1..(4*9)]of Byte
[AR1]    9: =($27,$Ø,$7Ø,$1,
[DK1]   1Ø:       $27,$Ø,$71,$1,
[GK1]   11:       $27,$Ø,$72,$1,
[LK1]   12:       $27,$Ø,$73,$1,
[PK1]   13:       $27,$Ø,$74,$1,
[TK1]   14:       $27,$Ø,$75,$1,
[WK1]   15:       $27,$Ø,$76,$1,
[ZK1]   16:       $27,$Ø,$77,$1,
[5U1]   17:       $27,$Ø,$78,$1);
[DV2]   18:       format_tabelle2:
[SF3]   19:       array[1..(4*9)]of Byte
[ØN1]   2Ø: =($26,$Ø,$1,$Ø,
[EC1]   21:       $26,$Ø,$2,$Ø,
[GC1]   22:       $26,$Ø,$3,$Ø,
```

```
[KC1]    23:       $26,$Ø,$4,$Ø,
[MC1]    24:       $26,$Ø,$5,$Ø,
[PC1]    25:       $26,$Ø,$6,$Ø,
[SC1]    26:       $26,$Ø,$7,$Ø,
[UC1]    27:       $26,$Ø,$8,$Ø,
[7K1]    28:       $26,$Ø,$9,$Ø);
[RT2]    29:       disk_parameter:
[U53]    3Ø:       array[1..11]of Byte
[AN2]    31: =($CF,$2,$25,$1,$9,$2A,
[WE2]    32:       $FF,$5Ø,$F6,$2Ø,$4);
[N86]    33:       var disk_p_save:array[1..11]of Byte;
[242]    34:       r: Registers;
[5W1]    35:       trak: Byte;
[XU2]    36:       int_1e: pointer;
[356]    37: procedure Format(form_tab: Pointer);
[MUØ]    38: begin
[M51]    39:   r.AH:=5;
[861]    4Ø:   r.AL:=9;
[G51]    41:   r.CL:=1;
[G51]    42:   r.DL:=Ø;
[YW1]    43:   r.CH:=trak;
[C51]    44:   r.DH:=Ø;
[SL3]    45:   r.ES:=Seg(form_tab^);
[UL3]    46:   r.BX:=Ofs(form_tab^);
[YR1]    47:   Intr($13,r);
[ZNØ]    48: end;
[T16]    49: procedure Write_Msg(sektor: Byte);
[E84]    5Ø: const Text: string[255]
[UM4]    51: =('Diese Diskette ist durch '+
[RT5]    52:       'das Programm D-SCHUTZ geschützt'+
[VS3]    53:       '(c) by Michael Humpl, '+
[953]    54:       '8264 Waldkraiburg');
[MUØ]    55: begin
[C51]    56:   r.AH:=3;
[861]    57:   r.AL:=9;
[S62]    58:   r.CL:=sektor;
[G51]    59:   r.DL:=Ø;
[YW1]    6Ø:   r.CH:=trak;
[C51]    61:   r.DH:=Ø;
[RL2]    62:   r.ES:=Seg(Text);
[TL2]    63:   r.BX:=Ofs(Text);
[YR1]    64:   Intr($13,r);
[ZNØ]    65: end;
[MUØ]    66: begin
[LC1]    67:   trak:=39;
[VH3]    68:   GetIntVec($1E,int_1e);
[R76]    69:   Move(Mem[Seg(int_1e^):Ofs(int_1e^)],
[FS2]    7Ø:   disk_p_save,11);
[PN3]    71:   Move(disk_parameter,
[7Ø6]    72:   Mem[Seg(int_1e^):Ofs(int_1e^)],11);
[Ø51]    73:   r.AH:=Ø;
[YR1]    74:   Intr($13,r);
[U94]    75:   Format(@format_tabelle1);
[DG2]    76:   Write_Msg(12Ø);
[VL1]    77:   Dec(trak);
[U36]    78:   Mem[Seg(int_1e^): Ofs(int_1e^)+3]:=Ø;
[Y94]    79:   Format(@format_tabelle2);
[Z52]    8Ø:   Write_Msg(4);
[L62]    81:   Write_Msg(9);
[733]    82:   Move(disk_p_save,
[DØ6]    83:   Mem[Seg(int_1e^): Ofs(int_1e^)],11);
[Ø51]    84:   r.AH:=Ø;
[YR1]    85:   Intr($13,r);
[KNØ]    86: end.
```

Listing 14. »d__schutz.pas« formatiert die Spuren 38 und 39 einer 360-KByte-Diskette

```
[V23]     1: Program disktest;
[KU3]     2: uses test_org,Dos,Crt;
[RW2]     3: var big:Longint;
[MUØ]     4:     begin
[DB1]     5:       ClrScr;
[295]     6:       if NOT test then Write(#7+'kein');
[ED3]     7:       WriteLn(' Orginal !');
[KNØ]     8:     end.
```

Listing 15. »disktest.pas« verwendet die Unit »test__org.pas«, um zu testen, ob Spuren 38 und 39 formatiert sind

```
[D94]     1: (Programm: test_org.pas
[H75]     2:  Funktion: Testet Kopierschutz
[LV5]     3:  Sprache: Turbo Pascal ab Version 5.5
[GM3]     4:  Autor:   Michael Humpl
[TX4]     5:  (c)1991 DMV Widuch GmbH & Co. KG}
[1K2]     6: unit Test_Org;
[SK1]     7: interface
[7W3]     8: function Test:Boolean;
[4L2]     9: implementation
[S62]    1Ø: uses Crt,Dos;
[VP3]    11: const disk_parameter:
[U53]    12:       array[1..11]of Byte
[AN2]    13: =($CF,$2,$25,$1,$9,$2A,
[WE2]    14:       $FF,$5Ø,$F6,$2Ø,$4);
[N86]    15:       var disk_p_save:array[1..11]of Byte;
[F72]    16:       r:Registers;
```

```
[US1]    17:       trak:Byte;
[1Ø2]    18:       ok:Boolean;
[5S2]    19:       int_1e:pointer;
[ND3]    2Ø:       buffer:string[255];
[ND4]    21:       rest:array[1..256]of Char;
[W92]    22:       Output:Text;
[ØH5]    23: procedure Read_Msg(sektor:Byte);
[FX3]    24: const Text:string[255]
[UM4]    25: =('Diese Diskette ist durch '+
[RT5]    26:       'das Programm D-SCHUTZ geschützt'+
[VC4]    27:       '(c) by Michael Humpl, 8264 '+
[NL2]    28:       'Waldkraiburg');
[MUØ]    29: begin
[851]    3Ø:   r.AH:=2;
[861]    31:   r.AL:=9;
[S62]    32:   r.CL:=sektor;
[G51]    33:   r.DL:=Ø;
[YW1]    34:   r.CH:=trak;
[C51]    35:   r.DH:=Ø;
[VV2]    36:   r.ES:=Seg(buffer);
[XV2]    37:   r.BX:=Ofs(buffer);
[YR1]    38:   Intr($13,r);
[R85]    39:   if buffer<>Text then ok:=false;
[ZNØ]    4Ø: end;
[7W3]    41: function Test:Boolean;
[MUØ]    42: begin
[HK1]    43:   ok:=True;
[LC1]    44:   trak:=39;
[NT1]    45:   buffer:='';
[VH3]    46:   GetIntVec($1E,int_1e);
[R76]    47:   Move(Mem[Seg(int_1e^):Ofs(int_1e^)],
[FS2]    48:   disk_p_save,11);
[PN3]    49:   Move(disk_parameter,
[7Ø6]    5Ø:   Mem[Seg(int_1e^):Ofs(int_1e^)],11);
[Ø51]    51:   r.AH:=Ø;
[YR1]    52:   Intr($13,r);
[H62]    53:   Read_Msg(12Ø);
[VL1]    54:   Dec(trak);
[PØ6]    55:   Mem[Seg(int_1e^):Ofs(int_1e^)+3]:=Ø;
[YX1]    56:   Read_Msg(4);
[DY1]    57:   Read_Msg(9);
[733]    58:   Move(disk_p_save,
[7Ø6]    59:   Mem[Seg(int_1e^):Ofs(int_1e^)],11);
[Ø51]    6Ø:   r.AH:=Ø;
[YR1]    61:   Intr($13,r);
[ØM2]    62:   if ok=false then
[KW4]    63:   Test:=false else Test:=True;
[ZNØ]    64: end;
[KNØ]    65: end.
```

Listing 16. »test__org.pas« kompilieren Sie zu einer Unit

Speichern inhomogener Datenmengen

Jeder, der »nur« einen PC betreibt, weiß, daß Berechnungen – vor allem von großen Feldern – eine Unmenge an Zeit verbrauchen. Wenn ein solches Programm läuft, können die Berechungen Stunden dauern. Was aber soll geschehen, wenn Sie in dieser Zeit plötzlich am PC arbeiten müssen? Entweder Sie brechen das Programm ab und beginnen später von neuem mit dem Programmablauf , oder Sie lassen alle Daten in eine Datei speichern. Wer mehrere Programme dieser Art einmal geschrieben hat, weiß, daß die Variablen, die Sie jeweils speichern wollen, meist verschieden sind.

Diese Gedanken legen nahe, nach einem Weg zu suchen, wie inhomogene Datenmengen zu speichern sind. »ds__vers.pas« (Listing 17) zeigt die Lösung auf. Zu beachten ist, daß Sie die Unit »dssave.pas« (Listing 18) als letzte einbinden. Die Unit enthält die Prozeduren »Save__DS« und »Load__DS«. Die Prozedur speichert das Datensegment zwischen »ende__ds« und »start__ds«. Hier wird die Tatsache genutzt, daß Konstanten am Beginn von »DS« (Datensegment) stehen und aktuelle Variable am Ende von »DS«. Der Speicherbereich wird in die Datei »save.ds« geschrieben und kann über »Load__DS« zurückgeladen werden. Diverse Fehlermeldungen sind in »Save__DS« und »Load__DS« enthalten, um Ihnen die richtige Handhabung von »save.ds« zu erleichtern.

Die hier aufgeführte Speicherung des »DS« hat den Nebeneffekt, daß Sie die Datei »save.ds« einfach mit einem entsprechenden Editor anzeigen können. In diesem Zusammenhang lernen Sie viel über die interne Datenverwaltung von Pascal. Bis jetzt nicht zu erklärende Fehler in der Variablenverwaltung können Sie somit aufdecken.

segment) stehen und aktuelle Variable am Ende von »DS«. Der Speicherbereich wird in die Datei »save.ds« geschrieben und kann über »Load_DS« zurückgeladen werden. Diverse Fehlermeldungen sind in »Save_DS« und »Load_DS« enthalten, um Ihnen die richtige Handhabung von »save.ds« zu erleichtern.

Die hier aufgeführte Speicherung des »DS« hat den Nebeneffekt, daß Sie die Datei »save.ds« einfach mit einem entsprechenden Editor anzeigen können. In diesem Zusammenhang lernen Sie viel über die interne Datenverwaltung

```
[YY3]    1: {Programm: ds_vers.pas
[7B6]    2:  Funktion: Sichert inhomogene Variable
[LV5]    3:  Sprache: Turbo Pascal ab Version 5.5
[EA3]    4:  Autor: Michael Humpl
[TX4]    5:  (c)1991 DMV Widuch GmbH & Co. KG)
[5U2]    6: uses Crt,dssave;
[457]    7: var versuch: array[1..1ØØ]of Byte;I: Byte;
[MUØ]    8:     begin
[DB1]    9:       ClrScr;
[XW2]   1Ø:       WriteLn('Start');
[1T1]   11:       Randomize;
[AA5]   12:       for I:=1 to 1ØØ do versuch[I]:=I;
[455]   13:       WriteLn('Daten vor Sicherung');
[5T2]   14:       for I:=1 to 1ØØ do
[8M3]   15:       Write(versuch[I],' ');
[XF1]   16:       WriteLn;
[223]   17:       save_ds(ende_ds);
[6V5]   18:       WriteLn('Daten mit Zufallswerten '+
[NE2]   19:       'beschreiben');
[5T2]   2Ø:       for I:=1 to 1ØØ do
[MUØ]   21:       begin
[UY3]   22:         versuch[I]:=Random(1ØØ);
[8M3]   23:         Write(versuch[I],' ');
[ZNØ]   24:       end;
[XF1]   25:       WriteLn;
[AN6]   26:      WriteLn('Datenregister restaurieren');
[9Ø3]   27:       load_ds(ende_ds);
[5T2]   28:       for I:=1 to 1ØØ do
[8M3]   29:       Write(versuch[I],' ');
[144]   3Ø:       WriteLn; WriteLn('Ende');
[KNØ]   31:     end.
```

Listing 17. Mit »ds_vers.pas« speichern Sie inhomogene Datenmengen ab

von Pascal. Bis jetzt nicht zu erklärende Fehler in der Variablenverwaltung können Sie somit aufdecken.

»ds_vers« speichert als erstes die Zahlen 1 bis 100 in ein Variablenfeld und zeigt dieses auf dem Bildschirm an. Anschließend speichern Sie über »Save_DS« die Variablen des Hauptprogramms. Nun wird das Variablenfeld mit Zufallszahlen gefüllt und angezeigt, die Variablen des Hauptprogramms zurückgeladen und das Variablenfeld angezeigt. Der ganze Vorgang ließe sich auch so programmieren, daß das Programm zwischen »Save_DS« und »Load_DS« abgebrochen und neu gestartet wird.

(Michael Humpl/et)

Automatisches Festplatten-Parken

»autopark.pas« (Listing 19) parkt alle im Computer befindlichen Festplatten, wenn nach dem letzten Tastendruck mehr als die vom Benutzer angegebene Zeit abgelaufen ist. Das speicherresidente Programm klinkt sich in den Tasaturinterrupt 9 und den Timerinterrupt 1Chex ein.

Autopark prüft als erstes, ob Sie eine gültige Wartezeit als Programmparameter eingegeben haben. Geben Sie keinen Programmparameter ein, oder übergeben Sie eine Wartezeit mit ungültigem Wert, so gibt Autopark eine entsprechende Meldung aus und beendet sich.

Als nächstes prüft Autopark, ob im Computer eine Festplatte vorhanden und ob diese auch betriebsbereit ist. Tritt bei diesem Test ein Fehler auf, wird eine entsprechende Meldung ausgegeben und das Programm beendet. Nun wird die Wartezeit im Hauptspeicher an der Position 0:194hex gespeichert. Wenn die Position 0:0195hex den Wert 1 enthält, haben Sie Autopark bereits installiert. Das Programm

```
[KM3]    1: {Programm: dssave.pas
[GP4]    2:  Funktion: Unit für ds_vers
[LV5]    3:  Sprache: Turbo Pascal ab Version 5.5
[EA3]    4:  Autor: Michael Humpl
[TX4]    5:  (c)1991 DMV Widuch GmbH & Co. KG)
[A42]    6: unit dssave;
[SK1]    7: interface
[274]    8: const start_ds: Char='S';
[F33]    9: var ende_ds: Char;
[4G5]   1Ø: procedure save_ds(var ende_ds);
[9E5]   11: procedure load_ds(var ende_ds);
[4L2]   12: implementation
[5H1]   13: uses Crt;
[U83]   14: procedure save_ds;
[WW1]   15: var f: file;
[GC1]   16: ch: Char;
[MUØ]   17: begin
[T13]   18:   Assign(f,'SAVE.DS');
[1A3]   19:   {$I-} Reset(f);{$I+}
[FØ3]   2Ø:   if IOResult=Ø then
[MUØ]   21:   begin
[KD1]   22:     Close(f);
[S36]   23:     WriteLn(#7+'Es wurde nach dem letzten'
[PR3]   24: +' Aufruf von Save_ds');
[TW5]   25:     WriteLn('kein Load_ds aufgerufen!');
[T16]   26:     WriteLn('Ist dies Beabsichtigt, so '+
[MT4]   27:     'drücken Sie bitte die Taste'+
[VF1]   28:     ' >> J <<.');
[FZ5]   29:     WriteLn('Jede andere Taste die Über'+
[1Y3]   3Ø:     'schreibung der alten '+
[2X3]   31:     'Variablensicherung!');
[222]   32:     ch:=ReadKey;
[L64]   33:     if UpCase(ch)='J' then Exit;
[ZNØ]   34:   end;
[CW3]   35:   {$I-} Rewrite(f,1);{$I+}
[T53]   36:   if IOResult<>Ø then
[MUØ]   37:   begin
[AW5]   38:     WriteLn('Die Datei >> SAVE.DS << kann'
[464]   39: +' nicht angelegt werden!');
[ØF6]   4Ø:     WriteLn('Das Datensegment wurde nicht'
[C92]   41: +' gesichert');
[N76]   42:     WriteLn('Bitte drücken Sie nach dem '+
[TF4]   43:     'Lesen dieser Meldung eine '+
[UF1]   44:     'Taste.');
[222]   45:     ch:=ReadKey;
[ØWØ]   46:     Exit;
[ZNØ]   47:   end;
[AF6]   48:   BlockWrite(f,Mem[DSeg: Ofs(start_ds)],
[SW4]   49:   Ofs(ende_ds)-Ofs(start_ds));
[KD1]   5Ø:   Close(f);
[ZNØ]   51: end;
[Z63]   52: procedure load_ds;
[WW1]   53: var f: file;
[GC1]   54: ch: Char;
[MUØ]   55: begin
[T13]   56:   Assign(f,'SAVE.DS');
[1A3]   57:   {$I-} Reset(f);{$I+}
[T53]   58:   if IOResult<>Ø then
[MUØ]   59:   begin
[4T5]   6Ø:     WriteLn(#7+'Die Datei >> SAVE.DS << '+
[1E5]   61:     'kann nicht geöffnet werden!');
[Ø87]   62:     WriteLn('Vor Aufruf von Load_DS muß '+
[KZ4]   63:     'Save_Ds ausgeführt worden '+
[Y61]   64:     'sein!');
[P76]   65:     WriteLn('Bitte nach dem Lesen dieser'+
[8H3]   66:     ' Meldung eine Taste '+
[NV1]   67:     'drücken.');
[222]   68:     ch:=ReadKey;
[ØWØ]   69:     Exit;
[ZNØ]   7Ø:   end;
[DXØ]   71:   {$I-}
[N76]   72:   BlockRead(f,Mem[DSeg: Ofs(start_ds)],
[SW4]   73:   Ofs(ende_ds)-Ofs(start_ds));
[M73]   74:   if IOResult<>Ø then;
[BXØ]   75:   {$I+}
[KD1]   76:   Close(f);
[GC1]   77:   Erase(f);
[ZNØ]   78: end;
[KNØ]   79: end.
```

Listing 18. Die Unit »dssave.pas« müssen Sie als letztes in »ds_vers.pas« einbinden

wird an dieser Stelle beendet. Andernfalls biegt Autopark die alten Interruptvektoren der Interrupts 9 und 1Chex auf die Interrupts 70hex und 71hex um. Dabei trägt Autopark die beiden Interruptprozeduren »NewInt1« und »NewInt9« als neue Interrupts 1Chex und 9 ein. In die Speichervariable »OldTimer» wird der aktuelle »BiosTimer« eingetragen.

Anschließend erhalten Sie die Meldung, daß die Installation erfolgreich war, womit Autopark speicherresident installiert ist.

Vor allem, wenn ein Computer längere Zeit eingeschaltet aber unbenutzt ist, besteht die Gefahr, daß sich der Schreib/Lesekopf der Festplatte über einer Datenspur befin-

det. Wenn jemand jetzt den Rechner zu stark bewegt, kann es geschehen, daß der Schreib/Lesekopf die Plattenoberfläche berührt. Dies muß zwar nicht unmittelbar einen Verlust der Daten dieser Datenspur bedeuten, doch können mit der Zeit plötzliche, unerwartete Schreib/Lesefehler auftreten. Viele Festplatten besitzen daher eine eigene Parkspur, die besonders beschichtet ist. Aus diesem Grunde sollte das System den Schreib-Lesekopf immer dann, wenn Sie die Festplatte länger nicht benutzen, parken. Wenn Sie Autopark einsetzen, ist die Festplatte geparkt, wenn der Computer länger als die vom Benutzer angegebene Wartezeit unbenutzt war. Abschließend und einschränkend muß allerdings angemerkt werden, daß bei einigen älteren Festplatten und Controllern die automatische Parkfunktion noch nicht vorgesehen ist.

(Michael Humpl/et)

Virensimulation in OOP-Technik

»demoviru.pas« (Listing 20) demonstriert anhand einiger auf dem Bildschirm tanzender Buchstaben das Prinzip der objektorientierten Programmierung.

Als erstes definieren Sie in Demoviru die Konstante »MaxVir«. Sie enthält die maximale Anzahl der angezeigten Gesichter (»Viren«). Werte zwischen 50 und 100 erweisen sich als gute Demonstration auf einem PC, der mit 8 MHz getaktet ist. Diesen Wert können Sie, solange die Größe des Arbeitsspeichers ausreicht, beliebig verändern. Zu beachten

```
[E84]    1: {Programm: autopark.pas
[SH7]    2:  Funktion: automatischen Festplatten-Parken
[LV5]    3:  Sprache: Turbo Pascal ab Version 5.5
[EA3]    4:  Autor: Michael Humpl
[TX4]    5:  (c)1991 DMV Widuch GmbH & Co. KG}
[S62]    6: uses Dos,Crt;
[1H2]    7: {$M 1ØØØØ,Ø,1ØØØØ}
[MZ4]    8: var OldInt1,OldInt9: pointer;
[DS4]    9:     BIOSTimer: Longint absolute
[N21]   1Ø:     $Ø: $46C;
[M23]   11:     OldTimer: Longint;
[MW1]   12:     Time: Byte;
[ØU2]   13:     Fehler: Integer;
[242]   14:     r: Registers;
[CX2]   15:     Install: Boolean;
[B92]   16:     anzahl: Byte;
[PN5]   17: procedure NewInt1(r2: Registers);
[8X1]   18: interrupt;
[MUØ]   19: begin
[SY1]   2Ø:   if(Time>Ø)AND
[994]   21: (Int(BIOSTimer-OldTimer)>
[VR2]   22: (Time*18.2*6Ø))then
[UH2]   23:   if Install then
[MUØ]   24:   begin
[Ø61]   25:     r.AH:=8;
[EC1]   26:     r.DL:=$8Ø;
[YR1]   27:     Intr($13,r);
[E52]   28:     anzahl:=r.DL;
[P31]   29:     repeat
[G81]   3Ø:       r.AH:=11;
[MH2]   31:       r.DL:=$7F+anzahl;
[Ø91]   32:       r.DH:=15;
[851]   33:       r.AL:=1;
[YR1]   34:       Intr($13,r);
[LZ1]   35:       Dec(anzahl);
[GK2]   36:     until anzahl=Ø;
[9R2]   37:     Install:=false;
[ZNØ]   38:   end;
[885]   39:   if OldInt1<>NIL then Intr($71,r2);
[ZNØ]   4Ø: end;
[6P5]   41: procedure NewInt9(r2: Registers);
[8X1]   42: interrupt;
[MUØ]   43: begin
[L23]   44:   Time:=Mem[$Ø:$194];
[XK3]   45:   OldTimer:=BIOSTimer;
[UH2]   46:   Install:=True;
[Y85]   47:   if OldInt9<>NIL then Intr($7Ø,r2);
[ZNØ]   48: end;
[MUØ]   49: begin
[L13]   5Ø:   checkbreak:=false;
[BV4]   51:   Val(ParamStr(1),Time,Fehler);
[FV3]   52:   if(Fehler>Ø)OR(Time>99)OR
[6Y1]   53: (Time<Ø)then
[MUØ]   54:   begin
[NT6]   55:    WriteLn(#7'Syntax: AUTOPARK [n], wobei'+
[G66]   56:     ' n die Minutenzahl bis Schutz ist!');
[T41]   57:     Halt(1);
[ZNØ]   58:   end;
[Ø61]   59:   r.AH:=8;
[EC1]   6Ø:   r.DL:=$8Ø;
[YR1]   61:   Intr($13,r);
[R42]   62:   if r.DL=Ø then
[MUØ]   63:   begin
[HV5]   64:     WriteLn(#7'Keine HD vorgefunden!');
[T41]   65:     Halt(1);
[HN3]   66:   end else anzahl:=r.DL;
[892]   67:   if r.AH<>Ø then
[MUØ]   68:   begin
[E86]   69:     WriteLn(#7'Fehler beim Ansprechen der'+
[8XØ]   7Ø:     ' HD!');
[T41]   71:     Halt(1);
[ZNØ]   72:   end;
[323]   73:   Mem[$Ø: $194]:=Time;
[U94]   74:   if Mem[$Ø: $195]=1 then Exit;
[MT3]   75:   GetIntVec($1C,OldInt1);
[FT3]   76:   GetIntVec($Ø9,OldInt9);
[XK3]   77:   OldTimer:=BIOSTimer;
[UH2]   78:   Install:=True;
[NV3]   79:   SetIntVec($1C,@NewInt1);
[SV3]   8Ø:   SetIntVec($Ø9,@NewInt9);
[ET3]   81:   SetIntVec($71,OldInt1);
[WT3]   82:   SetIntVec($7Ø,OldInt9);
[MB5]   83:   WriteLn('Autopark instaliert!');
[FE2]   84:   Mem[$Ø: $195]:=1;
[841]   85:   Keep(Ø);
[KNØ]   86: end.
```

Listing 19. »autopark.pas« parkt die Festplatte in Ihren schöpferischen Pausen

ist dabei, daß die Bewegung der »Viren« und damit auch die Vermehrung und Verminderung der Objekte immer langsamer wird, je größer die Anzahl der »Viren« ist, die Demoviru gleichzeitig bewegen muß. Als nächstes wird das Objekt »VIRUS« erzeugt, das Variablen für Aussehen sowie für Position und Bewegungsrichtung beinhaltet. Außerdem sind Prozeduren für das Initialisieren, Bewegen und Löschen enthalten. 1020: Die Technik der objektorientierten Programmierung ist für diesen Anwendungsfall besonders geeignet. Sie arbeitet mit einzelnen »Einheiten«, die zwar alle gleichartig aufgebaut sind und gleichartig agieren, aber trotzdem vollkommen unabhängig voneinander sind. Für jede dieser Einheiten richtet das Programm lediglich eine neue »Instanz« des Urtyps mit eigenen Variablen und ererbten Methoden ein. Alle Bildschirmausgaben schreibt Demoviru direkt in den Videospeicher. Das ist der Grund, warum Sie die Variablen »HSchirm« und »CSchirm« benötigen.

Damit die Virenanzeige unabhängig vom restlichen Programm ist, tragen Sie die Anzeige- und Bewegungshauptroutine »Timer« als neuen Interrupt 1Chex ein. Die Veränderung in ein TSR-Programm dürfte nur geringe Schwierigkeiten bereiten. Das Hauptprogramm gibt eine Meldung aus, welcher Sie entnehmen können, mit welchen Tasten Demoviru zu steuern ist.

Anschließend wird eine Schleife durchlaufen, welche auf das Drücken der Leertaste wartet. Da gleichzeitig die Prozedur »Timer« als Interrupt 1Chex eingetragen ist, welche das System also ungefähr 18mal in der Sekunde abfragt, wird die Bewegung der »Viren« auf dem Bildschirm simuliert. Nachdem Sie die Leertaste gedrückt haben, wird die Prozedur »Timer« aus dem Interrupt 1Chex gelöscht, und Sie erhalten wieder einen freien Bildschirm.

(Michael Humpl/et)

```
[U84]    1: {Programm: demoviru.pas
[MH4]    2:  Funktion: Virensimulation
[LV5]    3:  Sprache: Turbo Pascal ab Version 5.5
[EA3]    4:  Autor: Michael Humpl
[TX4]    5:  (c)1991 DMV Widuch GmbH & Co. KG}
[S62]    6: uses Crt,Dos;
[RN2]    7: const maxvir=5Ø;
[R13]    8: type VIRUS=object
[UB2]    9:       zeichen: Char;
[YP1]   1Ø:       x,y: Word;
[YP2]   11:       ou,lr: Shortint;
[ØE4]   12: procedure initialisieren;
[L33]   13: procedure bewegen;
[HD3]   14: procedure loeschen;
[ZNØ]   15: end;
[GN3]   16: var int_1ch: pointer;
[GC1]   17:     ch: Char;
```

Listing 20. »demoviru.pas« simuliert Viren in OOP-Technik

```
[P63]   18:      z1,maxausgabe: Word;
[E26]   19:      ausgabe: array[1..maxvir]of VIRUS;
[RC5]   2Ø:      HSchirm: array[1.. 25,1..8Ø,1..2]of
[UU3]   21:      Char absolute $BØØØ: $ØØØØ;
[Ø55]   22:      CSchirm: array[1..25,1..8Ø,1..2]of
[AV3]   23:      Char absolute $B8ØØ: $ØØØØ;
[AN5]   24: function ISchirm(x,y: Word): Char;
[MUØ]   25: begin
[LD3]   26:   if Mem[$Ø: $449]=7 then
[Y64]   27:   ISchirm:=HSchirm[y,x,1]
[ZRØ]   28:   else
[A84]   29:   ISchirm:=CSchirm[y,x,1];
[ZNØ]   3Ø: end;
[F96]   31: procedure OSchirm(x,y: Word;ch: Char);
[UV2]   32: var inhalt: Char;
[MUØ]   33: begin
[955]   34:   if ISchirm(x,y)IN[' ',#1,#2,#3,#4,#5,#6,
[Y52]   35:   #7]then begin
[LD3]   36:     if Mem[$Ø: $449]=7 then
[K53]   37:     HSchirm[y,x,1]:=ch
[ZRØ]   38:     else
[MC3]   39:     CSchirm[y,x,1]:=ch;
[ZNØ]   4Ø:   end;
[ZNØ]   41: end;
[VE4]   42: procedure VIRUS.loeschen;
[MUØ]   43: begin
[WL2]   44:   OSchirm(x,y,' ');
[ZNØ]   45: end;
[2P5]   46: procedure VIRUS.initialisieren;
[MUØ]   47: begin
[P31]   48:   repeat
[Ø54]   49:     zeichen:=Chr(Random(6)+1);
[8B2]   5Ø:     x:=Random(79)+1;
[HA2]   51:     y:=Random(24)+1;
[RR3]   52:   until ISchirm(x,y)=' ';
[Y72]   53:   ou:=Random(1);
[HØ3]   54:   if ou=Ø then ou:=-1;
[F72]   55:   lr:=Random(1);
[LZ2]   56:   if lr=Ø then lr:=-1;
[ZNØ]   57: end;
[XB4]   58: procedure VIRUS.bewegen;
[SU3]   59: var xa,ya,lra,oua: Word;
[MUØ]   6Ø: begin
[9C4]   61:   if((x+lr)>8Ø)OR((x+lr)<1)then
[MB1]   62:   lr:=-lr;
[GC4]   63:   if((y+ou)>25)OR((x+ou)<1)then
[SC1]   64:   ou:=-ou;
[MN4]   65:   if(ISchirm(x+lr,y+ou)=' ')then
[MUØ]   66:   begin
[WL2]   67:     OSchirm(x,y,' ');
[7B1]   68:     x:=x+lr;
[ZB1]   69:     y:=y+ou;
[LN3]   7Ø:     OSchirm(x,y,zeichen);
[3D1]   71:   end else
[MUØ]   72:   begin
[WL2]   73:     OSchirm(x,y,' ');
[832]   74:     xa:=x;ya:=y;
[YV2]   75:     lra:=lr;oua:=ou;
[7B1]   76:     x:=x+lr;
[ZB1]   77:     y:=y+ou;
[UD3]   78:     if(ISchirm(x-1,y)=' ')OR
[6X2]   79: (ISchirm(x+1,y)=' ')
[154]   8Ø:     then lr:=-lr else ou:=-ou;
```

```
[7B1]   81:     x:=x+lr;
[ZB1]   82:     y:=y+ou;
[ØN3]   83:     if ISchirm(x,y)<>' ' then
[MUØ]   84:     begin
[Z12]   85:       x:=xa;y:=ya;
[G13]   86:       lr:=-lra;ou:=-oua;
[ZNØ]   87:     end;
[ZNØ]   88:   end;
[ZNØ]   89: end;
[ØE4]   9Ø: procedure initialisieren;
[MUØ]   91: begin
[DB1]   92:   ClrScr;
[1T1]   93:   Randomize;
[9C2]   94:   maxausgabe:=1;
[ZNØ]   95: end;
[8T4]   96: procedure timer;interrupt;
[TU2]   97: var r: Registers;
[MUØ]   98: begin
[851]   99:   r.AH:=2;
[1S1]   1ØØ:  Intr($16,r);
[NU1]   1Ø1:  inline($FB);
[353]   1Ø2:  if(r.AL AND 1)>Ø then
[D74]   1Ø3:  if maxausgabe<maxvir then
[ØR2]   1Ø4:  Inc(maxausgabe);
[553]   1Ø5:  if(r.AL AND 2)>Ø then
[Z63]   1Ø6:  if maxausgabe>1 then
[MUØ]   1Ø7:  begin
[Ø75]   1Ø8:    ausgabe[maxausgabe].loeschen;
[VN2]   1Ø9:    Dec(maxausgabe);
[ZNØ]   11Ø:  end;
[P94]   111:  for z1:=1 to maxausgabe do
[SL3]   112:  ausgabe[z1].bewegen;
[KU1]   113:  inline($FA);
[ZNØ]   114: end;
[MUØ]   115: begin
[9S2]   116:  initialisieren;
[NM3]   117:  for z1:=1 to maxvir do
[XX4]   118:  ausgabe[z1].initialisieren;
[3R6]   119:  WriteLn('Aktuelle Anzahl an angezeigte'+
[7S1]   12Ø:  'n Viren: ');
[XF1]   121:  WriteLn;
[NG5]   122:  WriteLn('linker Shift-Taste  ->'+
[1X2]   123:  ' weniger Viren');
[NF5]   124:  WriteLn('rechter Shift-Taste ->'+
[292]   125:  ' mehr Viren');
[ZH4]   126:  WriteLn('Leertaste           ->'+
[251]   127:  ' Ende');
[9T3]   128:  GetIntVec($1C,int_1ch);
[2K3]   129:  SetIntVec($1C,@timer);
[P31]   13Ø:  repeat
[P31]   131:    repeat
[1Ø2]   132:      GotoXY(39,1);
[EC3]   133:      Write(maxausgabe: 4);
[K13]   134:    until KeyPressed;
[222]   135:    ch:=ReadKey;
[5W1]   136:  until ch=' ';
[1U3]   137:  SetIntVec($1C,int_1ch);
[P94]   138:  for z1:=1 to maxausgabe do
[PP3]   139:  ausgabe[z1].loeschen;
[1Ø2]   14Ø:  GotoXY(39,1);
[LP1]   141:  Write(Ø: 4);
[KNØ]   142: end.
```

Listing 20. »demoviru.pas« simuliert Viren in OOP-Technik

Anwendungen gesucht

Haben Sie eine schnelle und komfortable Textverarbeitung geschrieben? Oder eine Datenbank mit frei wählbaren Eingabemasken? Eventuell ein Zeichenprogramm? Wenn Ihr Programm mit professionellen Programmen konkurrieren kann, ist es mit Sicherheit zu umfangreich, um als Listing in der DOS International zu erscheinen. Zögern Sie deshalb nicht, schicken Sie uns unverbindlich die aktuelle Version Ihres Programms und einen Ausdruck der Dokumentation. Eventuell wird Ihr Programm der Verkaufsschlager im Rahmen der neuen DOS-Software-Schiene.

Schicken Sie Ihre Unterlagen an folgende Adresse:

Redaktion
DOS International,
Gruberstraße 46c
8011 Poing bei München

SCheck 3.0

Die nächste Generation

Das Abdrucken von Programmen mit Zeilennummern und Prüfsummen erleichtert das Abtippen und eventuelles Korrigieren der Listings. Ein Checksummer, der jetzt mit einer ansprechenden Benutzeroberfläche versehen ist, berechnet die korrekte Prüfsumme.

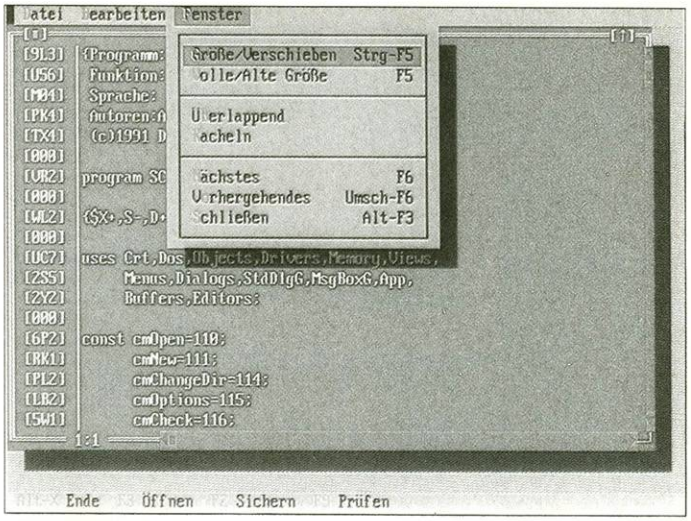


Bild 1. SCheck 3.0 mit grafischer Benutzeroberfläche

In der DOS International ist jede Zeile eines Programmlistings mit einer Prüfsumme und einer Zeilennummer versehen. Sind diese Zeilen länger als 55 Zeichen, entsteht ein Zeilenumbruch, die neue Zeile erhält weder Prüfsumme noch Zeilennummer. Diese Zeilenumbrüche dürfen Sie bei der Benutzung von SCheck 3.0 nicht mit eingeben, das heißt Sie schreiben einfach weiter. Tun Sie dies nicht, so berechnet der Checksummer für diese Programmzeile eine falsche Checksumme. Die vorangestellten Zeilennummern tippen Sie, außer bei Basic-Programmen, wenn sie ohne folgenden Doppelpunkt gedruckt sind, nicht mit ab – sie dienen lediglich der besseren Übersicht.

Die bisherigen Versionen des in der DOS International abgedruckten Programms SCheck hatten einige Nachteile bezüglich Handhabung und Geschwindigkeit. Mit dem neuen Turbo Pascal 6.0 und seiner Vision-Bibliothek gehört dies der Vergangenheit an (Bild 1).

Verwenden Sie den neuen Checksummer zum ersten Mal, fällt die neuartige Benutzeroberfläche auf, die dank Turbo Vision weitgehend dem SAA-Standard entspricht und mit der Maus bedient werden kann. Des weiteren verarbeitet er die Listings mit einer bisher nicht erreichten Geschwindigkeit. Zeigte der »alte« SCheck noch jede Zeile einzeln an und berechnete die zugehörige Prüfsumme, so tippen oder laden Sie nun zuerst das Listing in SCheck 3.0 und wählen anschließend den Menüpunkt »[D]atei [P]rüfen« an. SCheck 3.0 beginnt sogleich den Prüfvorgang und zeigt Ihnen den Namen, die aktuelle Zeilennummer und die bisher überprüfte Zeichenzahl an. Der Prüfvorgang ist sehr rasch beendet. Auf einem AT mit 16 MHz benötigt SCheck 3.0 für eine Selbstprüfung etwa vier Sekunden.

Die bei früheren SCheck-Versionen implementierte Methode, die Prüfsummen mit abzutippen und das Programm automatisch prüfen zu lassen, mußte beim neuen SCheck leider entfallen. Schuld daran ist der interne Aufbau des Vision-Editors.

Um Hexdump-Listings in ein lauffähiges Programm umzuwandeln, tippen Sie die Zeilen wie im Heft ohne Zeilennummer und Checksumme ab. Lassen Sie zwischen den einzelnen Zeichenpaaren immer ein Leerzeichen Abstand. Zulässige Zeichen in einem Hexdump-Listing sind »0« bis »9« sowie »A« bis »F«. Danach wählen Sie den Befehl »[D]atei [H]exdump umwandeln«, woraufhin SCheck 3.0 die Datei auf formale Fehler überprüft. Sobald der Checksummer einen Fehler gefunden hat, bricht er die Überprüfung ab und zeigt in einer Dialogbox an, in welcher Zeile der Fehler auftrat. Konnte SCheck 3.0 keinen Fehler finden, können Sie der Datei in einer Dialogbox noch einen Namen geben. Der Hexdump wird nun in einer lauffähigen Version auf Diskette oder Festplatte geschrieben.

Taste	Funktion
[F2]	Text speichern
[F3]	Datei laden
[F5]	Fenster zoomen
[F6]	nächstes Fenster aktivieren
[F9]	Prüfvorgang starten
[F10]	Menüleiste aktivieren
[Alt-F3]	aktives Fenster schließen
[Strg-F5]	Größe und Position des aktiven Fensters ändern
[Pos1]	Zeilenanfang
[Ende]	Zeilenende
[Strg-Pos1]	zum Bildschirmanfang
[Strg-Ende]	zum Bildschirmende
[Bild oben]	eine Seite nach oben
[Bild unten]	eine Seite nach unten
[Strg-Bild oben]	Dateianfang
[Strg-Bild unten]	Dateiende
[Strg-Einfg]	Block in Zwischenablage kopieren
[Strg-Entf]	Block löschen
[Umschalt-Einfg]	Block aus der Zwischenablage an der Cursorposition einfügen
[Umschalt-Entf]	Block in die Zwischenablage kopieren und aus dem Text entfernen

Tabelle 1. Die Tastenbelegung des Editors

Mehr Komfort durch den integrierten Editor

Mit SCheck 3.0 können Sie mehrere Listings gleichzeitig bearbeiten, Programmteile zwischen deren Fenstern hin und her kopieren, Programmteile verschieben und löschen. Im allgemeinen entspricht die Programmoberfläche der von Turbo Pascal 6.0 (Bild 2).

Der Editor entspricht in seiner Tastenbelegung weitgehend den Turbo-Pascal-Versionen ab 4.0. Einzelheiten entnehmen Sie bitte Tabelle 1. Es gibt zwei Wege, einen Block zu markieren:

- Sie benutzen die Maus, indem Sie die linke Maustaste gedrückt halten und den gewünschten Bereich markieren.
- Sie benutzen die Cursortasten und halten gleichzeitig eine Umschalt-Taste gedrückt.

Der Editor hat eine Kapazität von zirka 64 KByte, was auch für größere Programme reichen sollte. Die Menüpunkte des Editors entnehmen Sie den Tabellen 2 bis 4.

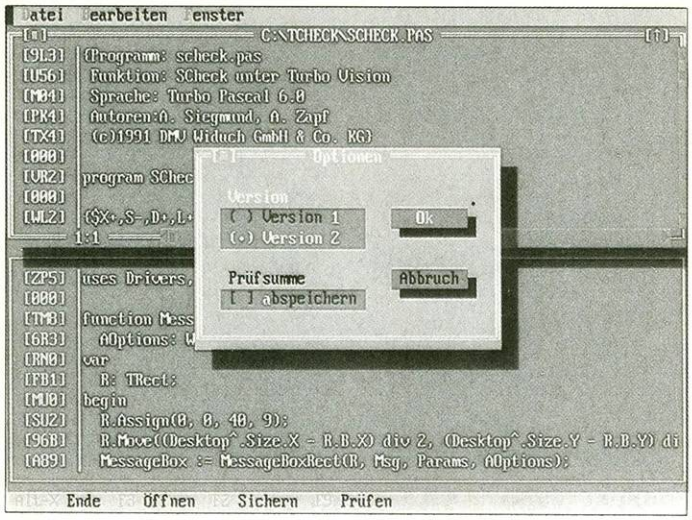


Bild 2. SCheck mit Turbo-Vision-Editor

Wenn Sie SCheck komplett in deutscher Sprache haben wollen, müssen Sie einige Änderungen am Quellcode der Turbo-Vision-Units »stddlg.pas« und »msgbox.pas« vornehmen. Die jeweils zu ändernden Zeilen finden Sie in »stddlg.chg« (Listing 2) und »msgbox.chg« (Listing 3). Am einfachsten ist folgende Vorgehensweise:
Laden Sie in einen beliebigen Editor den entsprechenden Quellcode. Dann ersetzen Sie die Zeilen im Quellcode der beiden Units durch die in Listing 2 beziehungsweise 3 angegebenen Zeilen. Speichern Sie anschließend die modifizierten Dateien unter dem Namen »stddlgg.pas« beziehungsweise »msgboxg.pas«. Ein Beispiel:
In Listing 2 steht folgende Zeile:

```
742 : D:='Verzeichnis';
```

Dies bedeutet: Ersetzen Sie im Quellcode der Unit »stddlg.pas« die Zeile 742 durch »D : = 'Verzeichnis'«. In »stddlgg.pas« entfallen einige Zeilen. Löschen Sie diese erst, nachdem Sie alle anderen Zeilen geändert haben, da sich sonst die Zeilennummern verschieben.

(A. Siegmund/A. Zapf/rm)

Menüpunkt	Bedeutung
Öffnen	lädt eine Datei von Diskette oder Festplatte in den Arbeitsspeicher
Neu	öffnet ein leeres Editorfenster mit dem Namen »noname«
Speichern	speichert die Datei im aktiven Fenster
Speichern unter	speichert die Datei im aktiven Fenster unter einem anderem Namen
Verzeichnis wechseln	wechselt das aktuelle Disketten- oder Festplattenverzeichnis
Optionen	Bei diesem Menüpunkt öffnet sich ein Dialogfenster, in dem Sie einstellen, nach welcher Version die Prüfsumme berechnet wird. Wer Listings aus der DOS International vor Oktober 1988 prüft, wählt hier Version 1. Später erschienene Listings benötigen die standardmäßig eingestellte Version 2. In dieser Dialogbox können Sie auch wählen, ob das Listing mit oder ohne Checksummen gespeichert wird.
Prüfen	beginnt mit dem Prüfen der Datei im aktiven Fenster
Hexdump umwandeln	wandelt ein Hexdump-Listing in ein lauffähiges Programm um
Über	Copyrightvermerk
Ende	Mit diesem Befehl können Sie SCheck ordnungsgemäß verlassen

Tabelle 2. Die Funktionen des Dateimenüs

Menüpunkt	Bedeutung
Wiederherstellen	macht die letzte Änderung am Text wieder rückgängig
Ausschneiden	verschiebt den markierten Block in die Zwischenablage
Kopieren	kopiert den markierten Block in die Zwischenablage
Einfügen	fügt einen zuvor in die Zwischenablage kopierten Block an der aktuellen Cursorposition ein
Löschen	löscht den markierten Programmteil

Tabelle 3. Die Funktionen des Bearbeitenmenüs

Menüpunkt	Bedeutung
Größe/Verschieben	läßt Änderungen in Größe und Position des aktiven Fensters zu
Volle/Alte Größe	stellt das Fenster in seinen größten beziehungsweise ursprünglichen Ausmaßen dar
Überlappend	ordnet die Fenster um eine Zeile versetzt übereinander auf dem Bildschirm an
Kacheln	ordnet die Fenster nebeneinander auf dem Bildschirm an
Nächstes	aktiviert das nächste Fenster
Vorhergehendes	aktiviert das vorherige Fenster
Schließen	schließt das aktive Fenster

Tabelle 4. Die Funktionen des Fenstermenüs

```
[9L3]    1: {Programm: scheck.pas
[U56]    2:  Funktion: SCheck unter Turbo Vision
[MØ4]    3:  Sprache: Turbo Pascal 6.Ø
[PK4]    4:  Autoren:A. Siegmund, A. Zapf
[TX4]    5:  (c)1991 DMV Widuch GmbH & Co. KG}
[ØØØ]    6:
[VR2]    7: program SCheck;
[ØØØ]    8:
[WL2]    9: {$X+,S-,D+,L+,V-}
[ØØØ]   1Ø:
[UC7]   11: uses Crt,Dos,Objects,Drivers,Memory,Views,
[2S5]   12:      Menus,Dialogs,StdDlgG,MsgBoxG,App,
[2Y2]   13:      Buffers,Editors;
[ØØØ]   14:
[6P2]   15: const cmOpen=11Ø;
[RK1]   16:       cmNew=111;
[PL2]   17:       cmChangeDir=114;
[LB2]   18:       cmOptions=115;
[5W1]   19:       cmCheck=116;
[YS2]   2Ø:       cmTransform=117;
[DZ1]   21:       cmAbout=118;
[NG3]   22:       cmOptionsChanged=2ØØ;
[A35]   23:       CheckChars='Ø123456789ABCDEFGHKL'
[UV2]   24:                  +'MNPRSTUVWXYZ';
[682]   25:       HeapSize=2Ø48;
[ØØØ]   26:
[STØ]   27: type
[ØP3]   28:  PCheckApp=^TCheckApp;
[W55]   29:  TCheckApp=object(TApplication)
[U43]   3Ø:    Constructor Init;
[494]   31:    Destructor Done;virtual;
[CK5]   32:    procedure HandleEvent(var Event:
[6W2]   33:                          TEvent);virtual;
[CE5]   34:    procedure InitMenuBar;virtual;
[BØ6]   35:    procedure InitStatusLine;virtual;
[B44]   36:    procedure Idle;virtual;
[7G5]   37:    procedure OutOfMemory;virtual;
[ZNØ]   38:  end;
[ØØØ]   39:
[EU3]   4Ø:  PCheckSum=^TCheckSum;
[FA4]   41:  TCheckSum=object(TObject)
[ØK2]   42:    Summe:PString;
[DX4]   43:    Constructor Init(s:string);
[494]   44:    Destructor Done;virtual;
[ZNØ]   45:  end;
[ØØØ]   46:
[CP5]   47:  PCheckInterior=^TCheckInterior;
[LZ4]   48:  TCheckInterior=object(TView)
[554]   49:    procedure Draw;virtual;
[ZNØ]   5Ø:  end;
[ØØØ]   51:
```

```
[PZ3]   52:  PCheckArea=^TCheckArea;
[844]   53:  TCheckArea=object(TView)
[813]   54:    List:PCollection;
[VY2]   55:    oldDelta:TPoint;
[LW2]   56:    Modified:Boolean;
[494]   57:    Destructor Done;virtual;
[554]   58:    procedure Draw;virtual;
[FK5]   59:    procedure SetState(AState:Word;
[H44]   60:            Enable:Boolean);virtual;
[8R5]   61:    procedure SetValue(Delta:TPoint;
[BK2]   62:                       Modif:Boolean);
[MG6]   63:    procedure NewList(AList:PCollection);
[ZNØ]   64:  end;
[ØØØ]   65:
[PK3]   66:  PNewEdit=^TNewEdit;
[S15]   67:  TNewEdit=object(TEditWindow)
[7K3]   68:    CheckArea:PCheckArea;
[XS2]   69:    Checked:Boolean;
[Z46]   70:    Constructor Init(var Bounds:TRect;
[7C3]   71:                     FileName:FNameStr);
[BC6]   72:    procedure HandleEvent(var Ev:TEvent);
[VF1]   73:              virtual;
[MR5]   74:    procedure SizeLimits(var Min,Max:
[FW2]   75:                         TPoint);virtual;
[W46]   76:    function GetTitle(AMaxLen:Integer):
[RC3]   77:                         TTitleStr;virtual;
[484]   78:    procedure Check;virtual;
[T45]   79:    procedure Transform;virtual;
[464]   80:    procedure Save;virtual;
[ND4]   81:    procedure SaveAs;virtual;
[CW4]   82:    procedure SaveFile;virtual;
[EW5]   83:    function ChkSum(s:string):string;
[ZNØ]   84:  end;
[ØØØ]   85:
[8K4]   86: PLineNumber=^TLineNumber;
[PB4]   87: TLineNumber=object(TView)
[GT1]   88: Line:Word;
[554]   89: procedure Draw;virtual;
[9P5]   90: procedure Update(var TLine:Word);
[ZNØ]   91: end;
[ØØØ]   92:
[Z34]   93: var CheckApp: TCheckApp;
[7P3]   94:     ClipWindow: PNewEdit;
[HV2]   95:     Optionen: record
[DV3]   96:       Version,CheckSum:Word
[ZNØ]   97:     end;
[ØØØ]   98:
[C95]   99: function ExecDialog(P:PDialog;
[AE3]  100:                     Data:pointer):Word;
[413]  101: var Result: Word;
[MUØ]  102: begin
[2Y2]  103:   Result:=cmCancel;
[ZG6]  104:   P:=PDialog(Application^.ValidView(P));
[B72]  105:   if P<>NIL then
[MUØ]  106:   begin
[ZV5]  107:     if Data<>NIL then P^.SetData(Data^);
[M55]  108:     Result:=DeskTop^.ExecView(P);
[FZ3]  109:     if(Result<>cmCancel)AND
[WV5]  110:       (Data<>NIL) then P^.GetData(Data^);
[4L2]  111:     Dispose(P,Done)
[ZNØ]  112:   end;
[W63]  113:   ExecDialog:=Result
[ZNØ]  114: end;
[ØØØ]  115:
[9E6]  116: function DoEditDialog(Dialog:Integer;
[V34]  117:                 Info:pointer):Word;far;
[MUØ]  118: begin
[WB2]  119:   case Dialog of
[ØK2]  120:     edOutOfMemory:
[ZA4]  121:       DoEditDialog:=MessageBox(
[6Ø6]  122:       'Nicht genug Speicher zum Arbeiten.',
[A94]  123:       NIL,mfError+mfOkButton);
[A52]  124:     edReadError:
[ZA4]  125:       DoEditDialog:=MessageBox(
[984]  126:       'Fehler beim Lesen von %s.',
[SH4]  127:       @Info,mfError+mfOkButton);
[7C2]  128:     edWriteError:
[ZA4]  129:       DoEditDialog:=MessageBox(
[4Z4]  130:       'Fehler beim Schreiben von %s.',
[SH4]  131:       @Info,mfError+mfOkButton);
[ZF2]  132:     edCreateError:
[ZA4]  133:       DoEditDialog:=MessageBox(
[VØ5]  134:       'Fehler beim Erstellen von %s.',
[SH4]  135:       @Info,mfError+mfOkButton);
[992]  136:     edSaveModify:
[ZA4]  137:       DoEditDialog:=MessageBox(
[YØ5]  138:       '%s wurde geändert. Sichern?',
[T96]  139:       @Info,mfConfirmation+mfYesNoCancel);
[XU2]  140:     edSaveUntitled:
[ZA4]  141:       DoEditDialog:=MessageBox(
[5N4]  142:       'Unbenannte Datei Sichern?',
[HW5]  143:       NIL,mfConfirmation+mfYesNoCancel);
[9H1]  144:     edSaveAs:
[T84]  145:       DoEditDialog:=ExecDialog(
[1B4]  146:       New(PFileDialog,Init('*.*',
[8W3]  147:       'Datei Speichern unter',
[C75]  148:       '~N~ame',fdOkButton,101)),Info)
[4MØ]  149:   end
[ZNØ]  150: end;
```

```
[ØØØ]  151:
[ZU6]  152: function OpenEditor(FileName:FNameStr;
[9M4]  153:              Visible:Boolean):PNewEdit;
[X22]  154: var P: PView;
[FB1]  155:     R: TRect;
[MUØ]  156: begin
[H45]  157:   if DeskTop^.Current<>NIL then
[MUØ]  158:   begin
[SL5]  159:     DeskTop^.Current^.GetBounds(R);
[4C3]  160:     Inc(R.A.X); Inc(R.A.Y)
[4MØ]  161:   end
[TU4]  162:   else DeskTop^.GetExtent(R);
[LZ6]  163:   P:=Application^.ValidView(New(PNewEdit,
[733]  164:   Init(R,FileName)));
[533]  165:   if NOT Visible then
[MUØ]  166:   begin
[7B1]  167:     P^.Hide;
[SG4]  168:     P^.Options:=P^.Options AND
[2S2]  169:                 NOT ofTileable;
[ZNØ]  170:   end;
[2F3]  171:   DeskTop^.Insert(P);
[YØ4]  172:   OpenEditor:=PNewEdit(P)
[ZNØ]  173: end;
[ØØØ]  174:
[BB2]  175: { TCheckApp }
[ØØØ]  176:
[RY4]  177: Constructor TCheckApp.Init;
[BY1]  178: var H: Word;
[FB1]  179:     R: TRect;
[3D4]  180:     DisCommands: TCommandSet;
[MUØ]  181: begin
[5S3]  182:   H:=PtrRec(HeapEnd).Seg-
[AC3]  183:      PtrRec(HeapPtr).Seg;
[2Z2]  184:   if H>HeapSize then
[L44]  185:     BufHeapSize:=H-HeapSize
[ZRØ]  186:   else
[7N2]  187:     BufHeapSize:=Ø;
[W62]  188:   InitBuffers;
[T63]  189:   TApplication.Init;
[9E6]  190:   DisCommands:=[cmSave,cmSaveAs,cmCut,
[HX3]  191:                 cmCopy,cmPaste,cmClear,
[FR4]  192:                 cmUndo,cmCheck,cmTransform,
[313]  193:                 cmTile,cmCascade];
[ES4]  194:   EditorDialog:=DoEditDialog;
[LH5]  195:   ClipWindow:=OpenEditor('',false);
[NW3]  196:   if ClipWindow<>NIL then
[MUØ]  197:   begin
[L85]  198:     Clipboard:=ClipWindow^.Editor;
[RF4]  199:     Clipboard^.CanUndo:=false;
[ZNØ]  200:   end;
[HX2]  201:   if LowMemory then
[GD4]  202:     DisCommands:=DisCommands+
[N73]  203:                  [cmOpen..cmAbout];
[TT2]  204:   with Optionen do
[MUØ]  205:   begin
[GZ3]  206:     Version:=1; CheckSum:=Ø
[ZNØ]  207:   end;
[465]  208:   DisableCommands(DisCommands);
[ZNØ]  209: end;
[ØØØ]  210:
[DF4]  211: Destructor TCheckApp.Done;
[MUØ]  212: begin
[N53]  213:   TApplication.Done;
[Z42]  214:   DoneBuffers
[ZNØ]  215: end;
[ØØØ]  216:
[BE7]  217: procedure TCheckApp.HandleEvent(var Event:
[BC1]  218:                                  TEvent);
[ØØØ]  219:
[LZ2]  220: procedure PrOpen;
[ZØ4]  221: var FileName: FNameStr;
[MUØ]  222: begin
[PA2]  223:   FileName:='*.*';
[FX4]  224:   if ExecDialog(New(PFileDialog,
[N24]  225:      Init('*.*','Datei Öffnen',
[SM4]  226:      '~N~ame',fdOpenButton,100)),
[P34]  227:      @FileName)<>cmCancel then
[LF4]  228:     OpenEditor(FileName,True);
[ZNØ]  229: end;
[ØØØ]  230:
[FW2]  231: procedure PrNew;
[MUØ]  232: begin
[573]  233:   OpenEditor('',True);
[ZNØ]  234: end;
[ØØØ]  235:
[8U3]  236: procedure PrChangeDir;
[MUØ]  237: begin
[NP4]  238:   ExecDialog(New(PChDirDialog,
[3F3]  239:              Init(cdNormal,Ø)),NIL)
[ZNØ]  240: end;
[ØØØ]  241:
[3P3]  242: procedure PrOptions;
[1W2]  243: var Dlg: PDialog;
[FB1]  244:     R: TRect;
[2F3]  245:     Labl,Control: PView;
```

Listing 1. »scheck.pas« – der neue Checksummer unter Turbo Vision

```
[YØ2]  246:     Dummy: Word;
[MUØ]  247: begin
[PR2]  248:   R.Assign(Ø,Ø,36,1Ø);
[DL4]  249:   New(Dlg,Init(R,'Optionen'));
[832]  25Ø:   with Dlg^do
[GP5]  251:     Options:=Options OR ofCentered;
[7R2]  252:     R.Assign(3,3,2Ø,5);
[X26]  253:     Control:=New(PRadioButtons,Init(R,
[164]  254:     NewSItem('Version ~1~',
[C85]  255:     NewSItem('Version ~2~',NIL))));
[2V4]  256:     PCluster(Control)^.Value:=1;
[ZR3]  257:     Dlg^.Insert(Control);
[3R2]  258:     R.Assign(3,2,11,3);
[CM5]  259:     Labl:=New(PLabel,Init(R,'Version',
[CR1]  26Ø:               Control));
[K23]  261:     Dlg^.Insert(Labl);
[HR2]  262:     R.Assign(3,7,2Ø,8);
[LL5]  263:     Control:=New(PCheckboxes,Init(R,
[NK4]  264:                  NewSItem('~a~bspeichern',
[941]  265:                  NIL)));
[YU4]  266:     PCluster(Control)^.Value:=Ø;
[ZR3]  267:     Dlg^.Insert(Control);
[RR2]  268:     R.Assign(3,6,13,7);
[G36]  269:     Labl:=New(PLabel,Init(R,'Prüfsumme',
[CR1]  27Ø:                           Control));
[K23]  271:     Dlg^.Insert(Labl);
[9S2]  272:     R.Assign(22,3,33,5);
[T86]  273:     Control:=New(PButton,Init(R,'~O~k',
[UR2]  274:                  cmOK,bfDefault));
[ZR3]  275:     Dlg^.Insert(Control);
[SS2]  276:     R.Assign(22,6,33,8);
[ØG6]  277:     Control:=New(PButton,Init(R,'Abbruch',
[Y73]  278:                  cmCancel,bfNormal));
[ZR3]  279:     Dlg^.Insert(Control);
[T24]  28Ø:     Dlg^.SelectNext(false);
[4B4]  281:     ExecDialog(Dlg,@Optionen);
[ZNØ]  282: end;
[ØØØ]  283:
[Z53]  284: procedure PrAbout;
[622]  285: var R: TRect;
[GT1]  286:     P: PDialog;
[MUØ]  287: begin
[6S2]  288:   R.Assign(Ø,Ø,48,21);
[5K3]  289:   P:=New(PDialog,Init(R,
[UK2]  29Ø:     'Über SCheck'));
[BN1]  291:   with P^do
[MUØ]  292:   begin
[GP5]  293:     Options:=Options OR ofCentered;
[UY1]  294:     R.Grow(-1,-1);
[GW1]  295:     Dec(R.B.Y,3);
[Ø34]  296:     Insert(New(PStaticText,
[CØ4]  297:       Init(R,#13+^C'SCheck'#13+#13+
[173]  298:            ^C'Version 1.Ø'#13+#13+
[9Y3]  299:            ^C'(c)1991 DMV Widuch GmbH'
[SØ2]  3ØØ:            +' & Co. KG'#13+
[BA3]  3Ø1:            #13+^C'Programm von'+
[CA3]  3Ø2:            '       Idee & Text von '
[NE4]  3Ø3:            +#13+#13+^C'Andreas Siegmund'
[D13]  3Ø4:            +'    Andreas Zapf    '
[SH4]  3Ø5:            +#13+#13+#13+^C'Checksummer'+#13
[S75]  3Ø6:            +#13+^C'für DOS International'
[ZW1]  3Ø7:            +' Listings'
[CDØ]  3Ø8:     )));
[Ø23]  3Ø9:     R.Assign(19,18,29,2Ø);
[WG6]  31Ø:     Insert(New(PButton,Init(R,'~O~K',cmOK,
[AW1]  311:                             bfDefault)))
[ZNØ]  312:   end;
[3S2]  313:   ExecDialog(P,NIL)
[ZNØ]  314: end;
[ØØØ]  315:
[VZ2]  316: procedure prTile;
[622]  317: var R: TRect;
[MUØ]  318: begin
[EV3]  319:   DeskTop^.GetExtent(R);
[WU2]  32Ø:   DeskTop^.Tile(R)
[ZNØ]  321: end;
[ØØØ]  322:
[2G3]  323: procedure prCascade;
[622]  324: var R: TRect;
[MUØ]  325: begin
[EV3]  326:   DeskTop^.GetExtent(R);
[593]  327:   DeskTop^.Cascade(R)
[ZNØ]  328: end;
[ØØØ]  329:
[ØØØ]  33Ø:
[MUØ]  331: begin
[DK5]  332:   TApplication.HandleEvent(Event);
[DT4]  333:   if Event.What=evCommand then
[MUØ]  334:   begin
[PE3]  335:     case Event.Command of
[8G2]  336:       cmOpen:PrOpen;
[U62]  337:       cmNew:PrNew;
[G84]  338:       cmChangeDir:PrChangeDir;
[2T3]  339:       cmOptions:PrOptions;
[YY2]  34Ø:       cmAbout:PrAbout;
[GF2]  341:       cmTile:prTile;
[GD3]  342:       cmCascade:prCascade;
[ZRØ]  343:       else
```

```
[ASØ]  344:         Exit
[ZNØ]  345:     end;
[5X2]  346:     ClearEvent(Event)
[4MØ]  347:   end
[ZNØ]  348: end;
[ØØØ]  349:
[SP5]  35Ø: procedure TCheckApp.InitMenuBar;
[622]  351: var R: TRect;
[MUØ]  352: begin
[782]  353:   GetExtent(R);
[F82]  354:   R.B.Y:=R.A.Y+1;
[S96]  355:   MenuBar:=New(PMenuBar,Init(R,NewMenu(
[616]  356:   NewSubMenu('~D~atei',hcNoContext,
[DV6]  357:   NewMenu(NewItem('Ö~f~fnen...','F3',kbF3,
[5K6]  358:   cmOpen,hcNoContext,NewItem('~N~eu','',
[BL4]  359:   kbNoKey,cmNew,hcNoContext,
[RH6]  36Ø:   NewItem('~S~peichern','F2',kbF2,cmSave,
[D72]  361:    hcNoContext,
[KC5]  362:   NewItem('Speichern ~u~nter...',
[5X6]  363:    '',kbNoKey,cmSaveAs,hcNoContext,NewLine(
[196]  364:   NewItem('~V~erzeichnis wechseln...',
[GX5]  365:    '',kbNoKey,cmChangeDir,hcNoContext,
[BZ5]  366:   NewItem('~O~ptionen...','',kbNoKey,
[7X3]  367:    cmOptions,hcNoContext,
[KB6]  368:   NewItem('~P~rüfen','F9',kbF9,cmCheck,
[D72]  369:    hcNoContext,
[SW5]  37Ø:   NewItem('~H~ex-Dump umwandeln...',
[126]  371:    '',kbNoKey,cmTransform,hcNoContext,
[DV6]  372:   NewItem('Ü~b~er...','',kbNoKey,cmAbout,
[D72]  373:    hcNoContext,
[BE1]  374:   NewLine(
[6M5]  375:   NewItem('~E~nde','Alt-X',kbNoKey,
[PA5]  376:    cmQuit,hcNoContext,NIL))))))))))))),
[2V6]  377:   NewSubMenu('~B~earbeiten',hcNoContext,
[4G1]  378:    NewMenu(
[K27]  379:   NewItem('~W~iederherstellen','',kbNoKey,
[4F3]  38Ø:    cmUndo,hcNoContext,
[BE1]  381:   NewLine(
[KR6]  382:   NewItem('~A~usschneiden','Umsch-Entf',
[K25]  383:    kbShiftDel,cmCut,hcNoContext,
[AX5]  384:   NewItem('~K~opieren','Strg-Einfg',
[325]  385:    kbCtrlIns,cmCopy,hcNoContext,
[776]  386:   NewItem('~E~infügen','Umsch-Einfg',
[DH5]  387:    kbShiftIns,cmPaste,hcNoContext,
[BE1]  388:   NewLine(
[PN5]  389:   NewItem('~L~öschen','Strg-Entf',
[C35]  39Ø:    kbCtrlDel,cmClear,hcNoContext,
[7G1]  391:    NIL))))))),
[LG6]  392:   NewSubMenu('~F~enster',hcNoContext,
[4G1]  393:   NewMenu(
[8L7]  394:   NewItem('~G~röße/Verschieben','Strg-F5',
[Y55]  395:    kbCtrlF5,cmResize,hcNoContext,
[HB6]  396:   NewItem('~V~olle/Alte Größe','F5',
[G94]  397:    kbF5,cmZoom,hcNoContext,
[BE1]  398:   NewLine(
[RD6]  399:   NewItem('Ü~b~erlappend','',kbNoKey,
[PR3]  4ØØ:    cmCascade,hcNoContext,
[7A5]  4Ø1:   NewItem('~K~acheln','',kbNoKey,
[DE3]  4Ø2:    cmTile,hcNoContext,
[BE1]  4Ø3:   NewLine(
[WE5]  4Ø4:   NewItem('~N~ächstes','F6',kbF6,
[PF3]  4Ø5:    cmNext,hcNoContext,
[SN6]  4Ø6:   NewItem('~V~orhergehendes','Umsch-F6',
[H15]  4Ø7:    kbShiftF6,cmPrev,hcNoContext,
[3N5]  4Ø8:   NewItem('~S~chließen','Alt-F3',
[7X4]  4Ø9:    kbAltF3,cmClose,hcNoContext,
[UP1]  41Ø:   NIL))))))))))),
[W51]  411:   NIL))))))
[ZNØ]  412: end;
[ØØØ]  413:
[5C6]  414: procedure TCheckApp.InitStatusLine;
[952]  415: var R:TRect;
[MUØ]  416: begin
[782]  417:   GetExtent(R);
[K82]  418:   R.A.Y:=R.B.Y-1;
[ØW3]  419:   New(StatusLine,Init(R,
[YD3]  42Ø:   NewStatusDef(Ø,$FFFF,
[556]  421:   NewStatusKey('~Alt-X~ Ende',kbAltX,
[4A1]  422:    cmQuit,
[CV6]  423:   NewStatusKey('~F3~ Öffnen',kbF3,cmOpen,
[UZ6]  424:   NewStatusKey('~F2~ Sichern',kbF2,cmSave,
[A47]  425:   NewStatusKey('~F9~ Prüfen',kbF9,cmCheck,
[VP4]  426:   NewStatusKey('',kbF1Ø,cmMenu,
[261]  427:   NIL))))),
[GPØ]  428:   NIL)
[D6Ø]  429: ))
[ZNØ]  43Ø: end;
[ØØØ]  431:
[DB4]  432: procedure TCheckApp.Idle;
[ØØØ]  433:
[EZ6]  434: function IsTileble(P:PView):Boolean;far;
[MUØ]  435: begin
[YP6]  436:   IsTileble:=P^.Options AND ofTileable<>Ø;
[ZNØ]  437: end;
[ØØØ]  438:
[MUØ]  439: begin
[P43]  44Ø:   TApplication.Idle;
[8L6]  441:   if DeskTop^.FirstThat(@IsTileble)<>NIL
```

```
[2RØ]  442:     then
[S96]  443:       EnableCommands([cmCheck,cmTransform,
[DZ2]  444:                       cmTile,cmCascade])
[ZRØ]  445:   else
[KH6]  446:     DisableCommands([cmCheck,cmTransform,
[T63]  447:                      cmTile,cmCascade]);
[ZNØ]  448: end;
[ØØØ]  449:
[7V5]  45Ø: procedure TCheckApp.OutOfMemory;
[MUØ]  451: begin
[5V5]  452:   MessageBox('Nicht genug Speicher.',
[F74]  453:               NIL,mfError+mfOkButton)
[ZNØ]  454: end;
[ØØØ]  455:
[KC2]  456: { TCheckSum }
[ØØØ]  457:
[YØ5]  458: Constructor TCheckSum.Init;
[MUØ]  459: begin
[PY2]  46Ø:   Summe:=NewStr(s);
[ZNØ]  461: end;
[ØØØ]  462:
[MH4]  463: Destructor TCheckSum.Done;
[MUØ]  464: begin
[AZ2]  465:   DisposeStr(Summe)
[ZNØ]  466: end;
[ØØØ]  467:
[MB3]  468: { TCheckInterior }
[ØØØ]  469:
[KB5]  47Ø: procedure TCheckInterior.Draw;
[8L3]  471: var buf: TDrawBuffer;
[H71]  472:     B: Byte;
[MUØ]  473: begin
[6E3]  474:   for B:=Ø to Size.Y do
[MUØ]  475:   begin
[FF5]  476:     MoveStr(buf,'       |',GetColor(1));
[8A3]  477:     WriteLine(Ø,B,8,1,buf)
[4MØ]  478:   end
[ZNØ]  479: end;
[ØØØ]  48Ø:
[3H2]  481: { TCheckArea }
[ØØØ]  482:
[ZN4]  483: Destructor TCheckArea.Done;
[MUØ]  484: begin
[FØ2]  485:   List^.Done;
[NS1]  486:   TView.Done
[ZNØ]  487: end;
[ØØØ]  488:
[1H4]  489: procedure TCheckArea.Draw;
[9Z1]  49Ø: var w: Word;
[3B2]  491:     B,Color: Byte;
[PL1]  492:     s: string;
[T32]  493:     P: PCheckSum;
[HW2]  494:     buf: TDrawBuffer;
[MUØ]  495: begin
[1C2]  496:   w:=oldDelta.Y;
[X62]  497:   with List^do
[MUØ]  498:   begin
[DPØ]  499:     B:=1;
[323]  5ØØ:     while(B<=Size.Y)do
[MUØ]  5Ø1:     begin
[GG2]  5Ø2:       if w<count then
[MUØ]  5Ø3:       begin
[YC1]  5Ø4:         P:=At(w);
[1A3]  5Ø5:         s:=' ['+P^.Summe^+'] '
[4MØ]  5Ø6:       end
[A72]  5Ø7:       else s:='       ';
[CN4]  5Ø8:       if(State AND sfDragging<>Ø)OR
[864]  5Ø9:         (State AND sfActive=Ø)then
[9B1]  51Ø:         Color:=1
[YA2]  511:       else Color:=2;
[ZS3]  512:       WriteStr(Ø,B-1,s,Color);
[M42]  513:       Inc(B); Inc(w)
[4MØ]  514:     end
[4MØ]  515:   end
[ZNØ]  516: end;
[ØØØ]  517:
[385]  518: procedure TCheckArea.SetState;
[MUØ]  519: begin
[G45]  52Ø:   TView.SetState(AState,Enable);
[A24]  521:   if(AState=sfDragging)OR
[GN3]  522:     (AState=sfActive)then
[BE1]  523:     DrawView
[ZNØ]  524: end;
[ØØØ]  525:
[X75]  526: procedure TCheckArea.SetValue;
[MUØ]  527: begin
[F54]  528:   if(Delta.Y<>oldDelta.Y)OR
[RH3]  529:     (Modif<>Modified)then
[MUØ]  53Ø:   begin
[7N5]  531:     oldDelta:=Delta;  Modified:=Modif;
[BE1]  532:     DrawView
[4MØ]  533:   end
[ZNØ]  534: end;
[ØØØ]  535:
[B25]  536: procedure TCheckArea.NewList;
[MUØ]  537: begin
[932]  538:   List:=AList
[ZNØ]  539: end;
```

```
[ØØØ]  54Ø:
[632]  541: { TNewEdit }
[ØØØ]  542:
[CS4]  543: Constructor TNewEdit.Init;
[622]  544: var R: TRect;
[6C1]  545:     P: PView;
[MUØ]  546: begin
[NV5]  547:   TEditWindow.Init(Bounds,FileName,
[F42]  548:                    wnNoNumber);
[ZL4]  549:   Editor^.GetBounds(Bounds);
[CW2]  55Ø:   Inc(Bounds.A.X,8);
[X25]  551:   Editor^.ChangeBounds(Bounds);
[SN3]  552:   R.Assign(1,1,9,Size.Y-1);
[745]  553:   P:=New(PCheckInterior,Init(R));
[R84]  554:   P^.GrowMode:=gfGrowHiY;
[ØN1]  555:   Insert(P);
[6W1]  556:   Dec(R.B.X,1);
[SP5]  557:   CheckArea:=New(PCheckArea,Init(R));
[LZ2]  558:   with CheckArea^do
[MUØ]  559:   begin
[1N3]  56Ø:     GrowMode:=gfGrowHiY;
[G53]  561:     Options:=Options;
[PØ3]  562:     Insert(CheckArea);
[PTØ]  563:     Hide;
[5Y5]  564:     NewList(New(PCollection,Init(Ø,Ø)))
[4MØ]  565:   end
[ZNØ]  566: end;
[ØØØ]  567:
[GK5]  568: procedure TNewEdit.HandleEvent;
[MUØ]  569: begin
[E64]  57Ø:   if Ev.What=evCommand then
[MUØ]  571:   begin
[F33]  572:     case Ev.Command of
[PB2]  573:       cmCheck:Check;
[G24]  574:       cmTransform:Transform;
[342]  575:       cmSave:Save;
[7V2]  576:       cmSaveAs:SaveAs;
[ZRØ]  577:       else
[MUØ]  578:       begin
[5Z4]  579:         TEditWindow.HandleEvent(Ev);
[ØWØ]  58Ø:         Exit;
[ZNØ]  581:       end;
[ZNØ]  582:     end;
[ZK2]  583:     ClearEvent(Ev);
[ZNØ]  584:   end;
[5Z4]  585:   TEditWindow.HandleEvent(Ev);
[AB3]  586:   if(CheckArea<>NIL)AND
[E24]  587:     (@self<>ClipWindow)then
[YC2]  588:     if Checked then
[XØ6]  589:       CheckArea^.SetValue(Editor^.Delta,
[TZ2]  59Ø:                           Editor^.Modified)
[ZNØ]  591: end;
[ØØØ]  592:
[1B5]  593: procedure TNewEdit.SizeLimits;
[MUØ]  594: begin
[AØ3]  595:   Min.X:=23; Min.Y:=5;
[MØ3]  596:   Max.X:=8Ø; Max.Y:=23
[ZNØ]  597: end;
[ØØØ]  598:
[4X4]  599: function TNewEdit.GetTitle;
[TX2]  6ØØ: var s: TTitleStr;
[MUØ]  6Ø1: begin
[VR5]  6Ø2:   s:=TEditWindow.GetTitle(AMaxLen);
[5D5]  6Ø3:   if s='Untitled' then s:='NONAME.';
[V32]  6Ø4:   GetTitle:=s
[ZNØ]  6Ø5: end;
[ØØØ]  6Ø6:
[N94]  6Ø7: procedure TNewEdit.Check;
[8Y1]  6Ø8: var B: Byte;
[D61]  6Ø9:     C: Char;
[DA3]  61Ø:     w,Bytes,Line: Word;
[8U2]  611:     s,Summe: string;
[KX1]  612:     P: PWindow;
[FB1]  613:     R: TRect;
[W76]  614:     ByteNumber,LineNumber: PLineNumber;
[3X2]  615:     Correct:Boolean;
[6D4]  616:  function FileName:string;
[LØ4]  617:  var s,ext,name: string;
[8R2]  618:  PEdit: PNewEdit;
[MUØ]  619:  begin
[R16]  62Ø:    PEdit:=PNewEdit(DeskTop^.Current);
[VA6]  621:    FSplit(PEdit^.GetTitle(Ø),s,name,ext);
[Y23]  622:    FileName:=name+ext
[ZNØ]  623:  end;
[ØØØ]  624:
[MUØ]  625: begin
[6R2]  626:   R.Assign(Ø,Ø,3Ø,9);
[MA5]  627:   P:=New(PDialog,Init(R,'Prüfen'));
[BN1]  628:   with P^do
[MUØ]  629:   begin
[9D1]  63Ø:     Flags:=Ø;
[GP5]  631:     Options:=Options OR ofCentered;
[EZ3]  632:     s:='Datei     : '+FileName;
[V34]  633:     R.Assign(3,2,Length(s)+3,3);
[EZ5]  634:     Insert(New(PStaticText,Init(R,s)));
```

Listing 1. »scheck.pas« – der neue Checksummer unter Turbo Vision

```
[RN2]  635:      s:='Zeichen   : ';
[344]  636:      R.Assign(3,4,Length(s)+3,5);
[EZ5]  637:      Insert(New(PStaticText,Init(R,s)));
[DS2]  638:      R.Assign(15,4,20,5);
[SG6]  639:      ByteNumber:=New(PLineNumber,Init(R));
[8C3]  640:      Insert(ByteNumber);
[LC2]  641:      s:='Zeile     : ';
[B44]  642:      R.Assign(3,6,Length(s)+3,7);
[EZ5]  643:      Insert(New(PStaticText,Init(R,s)));
[WS2]  644:      R.Assign(15,6,19,7);
[4F6]  645:      LineNumber:=New(PLineNumber,Init(R));
[M33]  646:      Insert(LineNumber)
[ZNØ]  647:    end;
[YP5]  648:    if CheckApp.ValidView(P)<>NIL then
[2F3]  649:     DeskTop^.Insert(P);
[TH2]  650:    with Editor^do
[MUØ]  651:    begin
[EB4]  652:      CheckArea^.List^.FreeAll;
[666]  653:      CheckArea^.List^.SetLimit(Limit.Y);
[EVØ]  654:      s:='';
[PRØ]  655:      w:=Ø;
[P81]  656:      Line:=Ø;
[KG1]  657:      Bytes:=Ø;
[TM5]  658:      while(w<=BufSize)AND(BufSize>Ø)do
[MUØ]  659:      begin
[1U2]  660:        if w<>CurPtr then
[MUØ]  661:        begin
[1Z1]  662:          Inc(Bytes);
[FH2]  663:          C:=Buffer^[w];
[UK4]  664:          if(C<>#13)AND(w<>BufSize)then
[UXØ]  665:            s:=s+C
[ZRØ]  666:          else
[MUØ]  667:          begin
[3N1]  668:            Inc(Line);
[4D4]  669:            LineNumber^.Update(Line);
[RN4]  670:            ByteNumber^.Update(Bytes);
[U42]  671:            s:=ChkSum(s);
[SH2]  672:            Correct:=True;
[GL4]  673:            if Optionen.CheckSum=1 then
[1N2]  674:            if s<>Summe then
[7R2]  675:            Correct:=false;
[XØ5]  676:            CheckArea^.List^.Insert(New(
[L43]  677:                        PCheckSum,Init(s)));
[5R1]  678:            s:=''; Inc(w)
[ZNØ]  679:          end;
[TYØ]  680:          Inc(w)
[4MØ]  681:        end
[ZY2]  682:        else Inc(w,GapLen)
[4MØ]  683:      end
[ZNØ]  684:    end;
[6Ø4]  685:    Checked:=True; P^.Close;
[VS2]  686:    CheckArea^.Show
[ZNØ]  687: end;
[ØØØ]  688:
[G45]  689: procedure TNewEdit.Transform;
[DA2]  690: var s: string;
[561]  691:     f: file;
[H71]  692:     B: Byte;
[V52]  693:     w,Line: Word;
[B74]  694:     Code: array[1..16]of Byte;
[472]  695:     Err: Integer;
[4A4]  696:     buf,TempBuf: PEditBuffer;
[XA2]  697:     Raus: Boolean;
[Y93]  698:     ByteCounter: Word;
[ØØØ]  699:
[EM3]  700:  procedure ReleaseMem;
[MUØ]  701:  begin
[TH2]  702:    with Editor^do
[MUØ]  703:    begin
[S16]  704:      FreeMem(buf,(BufSize-GapLen)DIV 3);
[134]  705:      FreeMem(TempBuf,BufLen)
[4MØ]  706:    end
[ZNØ]  707:  end;
[CB6]  708:  procedure Error(l:Longint;s:string);
[GC1]  709:  var Rec:
[Y11]  710:  record
[HA2]  711:    Long:Longint;
[YV1]  712:    St:PString
[ZNØ]  713:  end;
[MUØ]  714:  begin
[722]  715:    Rec.Long:=l;
[TY1]  716:    Rec.St:=@s;
[FV5]  717:    MessageBox('Falsches Zeilenformat'
[YK2]  718:      +' in Zeile %d!'+#13
[V25]  719:      +'%s!',@Rec,mfError+mfOkButton);
[NT1]  720:    ReleaseMem
[ZNØ]  721:  end;
[MUØ]  722: begin
[TH2]  723:   with Editor^do
[MUØ]  724:   begin
[TX5]  725:     buf:=MemAlloc((BufSize-GapLen)DIV 3);
[7D4]  726:     TempBuf:=MemAlloc(BufLen);
[FH6]  727:     Move(Buffer^[Ø],TempBuf^[Ø],CurPtr);
[U75]  728:     Move(Buffer^[CurPtr+GapLen],
[SV5]  729:          TempBuf^[CurPtr],BufLen-CurPtr);
[K32]  730:     w:=Ø; Line:=1;
[8T2]  731:     ByteCounter:=Ø;
[NT2]  732:     while w<BufLen do
[MUØ]  733:     begin
```

```
[ØR3]  734:       s:=''; B:=Ø; Raus:=false;
[P31]  735:       repeat
[8G3]  736:         if(TempBuf^[w]=#13)OR
[8F4]  737:           (w>=BufLen)then Raus:=True
[MY3]  738:         else s:=s+TempBuf^[w];
[EC4]  739:         if(Length(s)=3)OR(Raus)then
[MUØ]  740:         begin
[VC4]  741:           Val('$'+Copy(s,1,2),Code[B+1],
[7VØ]  742:               Err);
[344]  743:           if(Err<>Ø)OR((s[3]<>' ')AND
[9W2]  744:             (Length(s)=3))then
[MUØ]  745:           begin
[X52]  746:             if Err<>Ø then
[543]  747:               s:='Keine Hex-Zahl'
[ZRØ]  748:             else
[TY4]  749:               s:='Falsches Trennerzeichen';
[X82]  750:             Error(Line,s);
[ASØ]  751:             Exit
[ZNØ]  752:           end;
[SV1]  753:           Inc(B); s:=''
[ZNØ]  754:         end;
[561]  755:         Inc(w);
[X12]  756:       until Raus;
[RD4]  757:       if(B<>16)AND(w+1<BufLen)then
[MUØ]  758:       begin
[7Y1]  759:         Error(Line,
[NØ4]  760:               'Falsche Anzahl Zahlen');
[ASØ]  761:         Exit
[ZNØ]  762:       end;
[561]  763:       Inc(w);
[W45]  764:       Move(Code,buf^[(Line-1)*16],B);
[ZD3]  765:       Inc(ByteCounter,B);
[KF1]  766:       Inc(Line)
[ZNØ]  767:     end;
[K41]  768:     s:='*.*';
[FX4]  769:     if ExecDialog(New(PFileDialog,
[8G1]  770:        Init('*.*',
[SL4]  771:          'Hex-Dump speichern unter',
[E55]  772:          '~N~ame',fdOkButton,101)),@s)<>
[W52]  773:          cmCancel then
[MUØ]  774:     begin
[YW1]  775:       Assign(f,s);
[Y22]  776:       ReWrite(f,1);
[FE5]  777:       BlockWrite(f,buf^,ByteCounter);
[TM2]  778:       System.Close(f)
[4MØ]  779:     end
[ZNØ]  780:   end;
[NT1]  781:   ReleaseMem
[ZNØ]  782: end;
[ØØØ]  783:
[C74]  784: procedure TNewEdit.Save;
[MUØ]  785: begin
[GL4]  786:   if Optionen.CheckSum=1 then
[MUØ]  787:   begin
[LR5]  788:     if Editor^.FileName='' then  SaveAs
[TC2]  789:     else SaveFile;
[4MØ]  790:   end
[VY2]  791:   else Editor^.Save
[ZNØ]  792: end;
[ØØØ]  793:
[9K4]  794: procedure TNewEdit.SaveAs;
[MUØ]  795: begin
[GL4]  796:   if Optionen.CheckSum=1 then
[TH2]  797:   with Editor^do
[MUØ]  798:   begin
[X44]  799:     if DoEditDialog(edSaveAs,
[P34]  800:        @FileName)<>cmCancel then
[RN4]  801:       FileName:=FExpand(FileName);
[TD4]  802:     Message(@self,evBroadCast,
[YC3]  803:             cmUpdateTitle,NIL);
[G72]  804:     self.SaveFile
[4MØ]  805:   end
[AF3]  806:   else Editor^.SaveAs
[ZNØ]  807: end;
[ØØØ]  808:
[WX4]  809: procedure TNewEdit.SaveFile;
[CØ3]  810: var s,Sum: string;
[K81]  811:     w: Word;
[561]  812:     f: file;
[VT1]  813:     Txt: Text;
[4M3]  814:     BackupName: FNameStr;
[RL1]  815:     D: DirStr;
[KV1]  816:     N: NameStr;
[YM1]  817:     E: ExtStr;
[D61]  818:     C: Char;
[K12]  819:     Bo: Boolean;
[FW1]  820:     P: pointer;
[MUØ]  821: begin
[DXØ]  822:   {$I-}
[TH2]  823:   with Editor^do
[MUØ]  824:   begin
[3K1]  825:     Bo:=True;
[CR4]  826:     if FSearch(FileName,'')<>'' then
[MUØ]  827:     begin
[C52]  828:       P:=@FileName;
[HM2]  829:       Bo:=MessageBox(
[NX4]  830:           'Die Datei %s wird ersetzt!',@P,
[UE4]  831:           mfConfirmation+mfOkCancel)
[XN1]  832:         <>cmCancel
```

```
[ZNØ]  833:      end;
[GH1]  834:      if Bo then
[MUØ]  835:      begin
[2Y2]  836:        if EditorFlags AND
[BK3]  837:           efBackupFiles<>Ø then
[MUØ]  838:        begin
[7V3]  839:          FSplit(FileName,D,N,E);
[3P3]  84Ø:          BackupName:=D+N+'.BAK';
[KG3]  841:          Assign(f,BackupName);
[GC1]  842:          Erase(f);
[Y53]  843:          Assign(f,FileName);
[ZE3]  844:          Rename(f,BackupName);
[XZ1]  845:          InOutRes:=Ø
[ZNØ]  846:        end;
[KG3]  847:        Assign(Txt,FileName);
[AC2]  848:        ReWrite(Txt);
[T53]  849:        if IOResult<>Ø then
[HS4]  85Ø:          DoEditDialog(edCreateError,
[CM1]  851:                       @FileName)
[ZRØ]  852:        else
[MUØ]  853:        begin
[2N1]  854:          w:=Ø; s:='';
[8E3]  855:          while w<=BufSize do
[MUØ]  856:          begin
[1U2]  857:            if w<>CurPtr then
[MUØ]  858:            begin
[FH2]  859:              C:=Buffer^[w];
[654]  86Ø:              if(C=#13)OR(w=BufSize)then
[MUØ]  861:              begin
[RP3]  862:                s:='['+ChkSum(s)+'] '+s;
[3T2]  863:                WriteLn(Txt,s);
[5R1]  864:                s:=''; Inc(w)
[4MØ]  865:              end
[BY1]  866:              else s:=s+C;
[TYØ]  867:              Inc(w)
[4MØ]  868:            end
[ZY2]  869:            else Inc(w,GapLen)
[ZNØ]  87Ø:          end;
[P63]  871:          System.Close(Txt);
[8P2]  872:          Modified:=false
[4MØ]  873:        end
[4MØ]  874:      end
[4MØ]  875:    end
[BXØ]  876:    {$I+}
[ZNØ]  877: end;
[ØØØ]  878:
[EE4]  879: function TNewEdit.ChkSum;
[8Y1]  88Ø: var B: Byte;
[K81]  881:     w: Word;
[U12]  882:     St: string;
[MUØ]  883: begin
[MØ5]  884:   while(s[1]=' ')AND(Length(s)>Ø)do
[RA3]  885:   System.Delete(s,1,1);
[XA6]  886:   while(s[Length(s)]=' ')AND(Length(s)>Ø)do
[ØN4]  887:   System.Delete(s,Length(s),1);
[PRØ]  888:   w:=Ø;
[3W3]  889:   for B:=1 to Length(s)do
[MUØ]  89Ø:   begin
[874]  891:     case Optionen.Version of
[SH5]  892:       Ø:w:=w+((B AND 7)*Ord(UpCase(s[B])));
[ZR2]  893:       1:w:=w+((B AND 3)+1)*
[R13]  894:         Ord(UpCase(s[B]));
[4MØ]  895:     end
[ZNØ]  896:   end;
[P31]  897:   St:='';
[NF2]  898:   for B:=1 to 3 do
[MUØ]  899:   begin
[585]  9ØØ:     if w>MaxInt then Dec(w,MaxInt+1);
[D16]  9Ø1:     St:=St+Copy(CheckChars,(w AND 31)+1,1);
[EP1]  9Ø2:     w:=w DIV 32
[ZNØ]  9Ø3:   end;
[NT1]  9Ø4:   ChkSum:=St
[ZNØ]  9Ø5: end;
[ØØØ]  9Ø6:
[6X2]  9Ø7: { TLineNumber }
[ØØØ]  9Ø8:
[PU4]  9Ø9: procedure TLineNumber.Draw;
[DA2]  91Ø: var s: string;
[H71]  911:     B: Byte;
[MUØ]  912: begin
[K12]  913:   Str(Line,s);
[G53]  914:   while Length(s)<5 do
[341]  915:   s:=s+' ';
[CG2]  916:   B:=GetColor(1);
[6M2]  917:   WriteStr(Ø,Ø,s,2)
[ZNØ]  918: end;
[ØØØ]  919:
[V35]  92Ø: procedure TLineNumber.Update;
[MUØ]  921: begin
[YP3]  922:   Line:=TLine; DrawView
[ZNØ]  923: end;
[ØØØ]  924:
[MUØ]  925: begin
[FD2]  926:   CheckApp.Init;
[882]  927:   CheckApp.Run;
[U62]  928:   CheckApp.Done
[KNØ]  929: end.
```

Listing 1. »scheck.pas« – der neue Checksummer unter Turbo Vision

```
[ZM5]    1: Patches zur Unit »stddlg.pas«
[ØØØ]    2:
[VS2]    3:  11: unit StdDlgG;
[Z35]    4:  684: if P = nil then MessageBox(
[KX3]    5:       'Zu viele Dateien.', nil,
[C14]    6:       mfOkButton + mfWarning);
[X53]    7:  712: D: String[11];
[5Z5]    8:  719: Params: array[Ø..6] of LongInt;
[TE3]    9:  722: sDirectoryLine =
[3G4]   1Ø:      ' %-12s %-11s %2d. %3s %4d  %2d:%Ø2d';
[CD2]   11:  723: sFileLine =
[YS4]   12:      ' %-12s  %-1Ød %2d. %3s %4d  %2d:%Ø2d';
[7E6]   13:  724: Month: array[1..12] of String[3] =
[AB5]   14:       ('Jan','Feb','Mär','Apr','Mai','Jun',
[L65]   15:       'Jul','Aug','Sep','Okt','Nov','Dez');
[PF3]   16:  742: D := 'Verzeichnis';
[ØØØ]   17:
[KR6]   18:  Die Zeilen 751 bis 763 müssen vollständig
[UU3]   19:  ersetzt werden durch :
[ØØØ]   2Ø:
[794]   21:  751: Params[2] := Time.Day;
[8M4]   22:  752: Params[3] := LongInt(@m);
[ZC4]   23:  753: Params[4] := Time.Year;
[4F4]   24:  754: Params[5] := Time.Hour;
[R94]   25:  755: Params[6] := Time.Min;
[WL5]   26:  756: FormatStr(Str, FmtId, Params);
[1G4]   27:  757: MoveStr(B, Str, Color);
[N55]   28:  758: WriteLine(Ø, 1, Size.X, 1, B);
[ØØØ]   29:
[7V4]   3Ø:  Die Zeilen 759 bis 763 entfallen
[ØØØ]   31:
[W73]   32:  82Ø: R.Assign(2,5,1Ø,6);
[ØN5]   33:  821: Control := New(PLabel, Init(R,
[PY3]   34:       '~D~ateien', FileList));
[UC3]   35:  824: R.Assign(35,3,48,5);
[4F5]   36:  828: Insert(New(PButton, Init(R,
[ER4]   37:       '~L~aden', cmFileOpen, Opt)));
[RE5]   38:  834: Insert(New(PButton, Init(R,
[2H4]   39:       '~O~K', cmFileOpen, Opt)));
[AE5]   4Ø:  84Ø: Insert(New(PButton, Init(R,
[846]   41:       '~E~rsetzen',cmFileReplace, Opt)));
[2F5]   42:  846: Insert(New(PButton, Init(R,
[8L5]   43:       '~E~ntfernen',cmFileClear, Opt)));
[DE5]   44:  85Ø: Insert(New(PButton, Init(R,
[SØ5]   45:       'Abbruch', cmCancel, bfNormal)));
[XE5]   46:  854: Insert(New(PButton, Init(R,
[7A4]   47:       'Hilfe',cmHelp, bfNormal)));
[XC6]   48: 1Ø18: MessageBox('Ungültiges Laufwerk'
[KF3]   49:         +' oder Verzeichnis.',
[WH4]   5Ø:         nil, mfError + mfOkButton);
[GS6]   51: 1Ø59: MessageBox('Ungültiger Dateiname.',
[WH4]   52:         nil, mfError + mfOkButton);
[576]   53: 1Ø99: DrivesS: String[9] = 'Laufwerke';
[753]   54: 1275: TDialog.Init(R,
[6Ø4]   55:       'Verzeichnis wechseln');
[HR3]   56: 1282: R.Assign(2, 2, 18, 3);
[VB4]   57: 1283: Control := New(PLabel,
[LY4]   58:       Init(R, '~V~erzeichnisname',
[1Z1]   59:       DirInput));
[HS3]   6Ø: 1295: R.Assign(2, 5, 19, 6);
[KN5]   61: 1296: Control := New(PLabel, Init(R,
[TE5]   62:       'Verzeichnis-~B~aum', DirList));
[LU3]   63: 1299: R.Assign(35, 6, 46, 8);
[LZ5]   64: 13ØØ: OkButton := New(PButton, Init(R,
[H84]   65:       '~O~K', cmOK, bfDefault));
[BE6]   66: 13Ø3: ChDirButton := New(PButton, Init(R,
[ME4]   67:       '~W~echseln', cmChangeDir,
[HV1]   68:       bfNormal));
[U55]   69: 13Ø6: Insert(New(PButton, Init(R,
[Z16]   7Ø:       '~Z~urück', cmRevert, bfNormal)));
[S55]   71: 131Ø: Insert(New(PButton, Init(R,
[TZ4]   72:       '~H~ilfe', cmHelp, bfNormal)));
[B97]   73: 1415: MessageBox('Ungültiges Verzeichnis.',
[WH4]   74:         nil, mfError + mfOkButton);
```

Listing 2. Die Liste mit den zu modifizierenden Zeilen in »stddlg.pas«

```
[3N5]    1: Patches zur Unit »msgbox.pas«
[ØØØ]    2:
[HU2]    3:  11: Unit MsgBoxG;
[4Ø7]    4:  82: ButtonName: array[Ø..3] of string[7] =
[3A6]    5:       ('~J~a', '~N~ein', '~O~K', 'Abbruch');
[MF6]    6:  86: Titles: array[Ø..3] of string[11] =
[AG5]    7:       ('Warnung','Fehler','Information',
[7T2]    8:        'Bestätigung');
[TL3]    9: 1Ø9: R.Assign(Ø, Ø, 11, 2);
[WV6]   1Ø: 157: Insert(New(PButton, Init(R, '~O~K',
[A33]   11:              cmOk, bfDefault)));
[457]   12: 159: Insert(New(PButton, Init(R, 'Abbruch',
[TE3]   13:              cmCancel, bfNormal)));
```

Listing 3. Die Liste mit den zu modifizierenden Zeilen in »msgbox.pas«

Compilerbau

Compiler für Stapeldateien

Stapeldateien lassen sich zwar schnell schreiben, sind jedoch extrem langsam in ihrer Abarbeitung. Mit dem hier vorgestellten Compiler Batch__It wandeln Sie Stapeldateien in eigenständige, ausführbare Programme um, die zudem mit der Geschwindigkeit von Hochsprachenprogrammen ablaufen.

Mit jedem Rechner, der mit dem Betriebssystem MS-DOS oder DR-DOS ausgeliefert wird, steht Ihnen auch eine kleine betriebssysteminterne Programmiersprache zur Verfügung: die sogenannte Batch-Sprache. Deren Sprachumfang erlaubt es, Befehle des Betriebssystems auf einfache Weise nacheinander auszuführen. Da dies dem Abarbeiten eines Stapels gleicht, nennt man die Vorgehensweise Stapelverarbeitung. Mit ihrem rudimentären Sprachumfang umfaßt die Batch-Sprache die wichtigsten Merkmale, die eine Programmiersprache bieten muß. Trotzdem ist diese Sprache so leicht zu erlernen, daß auch ein Anfänger nach kurzer Zeit wirkungsvolle Stapeldateien entwickeln kann. Kenntnisse über eine »ausgewachsene« Hochsprache benötigt er dazu nicht.

Der Fortgeschrittene verwendet diese Sprache zur Realisierung von Anwendungen, deren Entwicklungsaufwand in einer der üblichen Programmiersprachen unverhältnismäßig höher wäre und dem Nutzen nicht entsprechen würde. So manches Beispiel aus der täglichen Praxis zeigt nicht nur die Leistungsstärke dieser Batch-Sprache, sondern macht auch deren flexiblen Einsatz deutlich.

Den Vorteilen leichte Erlernbarkeit und Flexibilität stand bisher der Nachteil der langsamen Ausführung von Stapeldateien gegenüber. Doch mit dem Einsatz von Batch__It entfällt auch dieses Manko, da die Ausführungsgeschwindigkeit von kompilierten Stapeldateien denen von Hochsprachenprogrammen kaum nachsteht.

Der vorliegende Compiler für Stapeldateien wurde in der Hochsprache C geschrieben und weist Prinzipien und Merkmale auf, die jedem ernsthaften Compiler zugrunde liegen sollten. Durch den recht einfachen Aufbau der einzelnen Module verstehen auch Anfänger der Sprache C Struktur und Funktionsweise von Batch__It. Sie erweitern Batch__It somit ohne großen Aufwand und schaffen auf diese Weise vielfältige Tools zur Vergrößerung des Sprachumfangs der Batch-Sprache.

In der Vergangenheit wurden zwar schon mehrere Versuche unternommen, einen Compiler für Stapeldateien zu realisieren, jedoch waren diese meist in Assembler geschrieben – sehr zum Nachteil derer, die ihre Programme in einer Hochsprache entwickeln. Durch den nicht gerade geringen Umfang und die Komplexität, die einem Compiler von Natur aus anhaftet, blieb sogar den Assembler-Kundigen der Zugang zu den Prinzipien der Compilerentwicklung meistens verwehrt. Die Unübersichtlichkeit des langen Assembler-Listings verschleierte fast gänzlich die Erkenntnisse über die eigentliche Arbeitsweise eines Compilers.

Der hier vorgestellte Compiler für Stapeldateien verwandelt Stapeldateien nicht nur in ausführbare Programme, sondern verleiht diesen auch eine für die Sprache ungewohnte Geschwindigkeit. Dieses Phänomen liegt nicht zuletzt in der unterschiedlichen Arbeitsweise von Interpreter und Compiler begründet.

Im Normalfall verarbeitet der Kommandointerpreter von MS-DOS eine Stapeldatei ähnlich, als würden Sie die Kommandos nacheinander eintippen. Allgemein läuft die Interpretation eines Programms folgendermaßen ab: Zunächst wird der Interpreter selbst in den Hauptspeicher geladen. Er stellt dem Programm seine Ablaufumgebung zur Verfügung und muß sich demnach auch während der eigentlichen Programmausführung im Hauptspeicher befinden. Dadurch steht dem Programm zur Laufzeit entsprechend weniger Arbeitsspeicher (abhängig von der Größe des Interpreters) zur Verfügung. Bei GW-Basic veranlassen Sie das Laden des Interpreters und Einrichten der sogenannten Laufzeitumgebung durch den Aufruf des Basic-Interpreters.

Zur Abarbeitung von Stapeldateien müssen Sie in der Regel keinen Interpreter laden, da sich der Kommandointerpreter von MS-DOS fast immer im Hauptspeicher befindet. Falls ein Programm ihn überschrieben hat, lädt sich der Kommandointerpreter sogar selbst nach.

Prinzipien und Arbeitsweise von Interpreter und Compiler

Vom geladenen Interpreter aus wird nun das abzuarbeitende Programm gestartet. Bei jedem Durchlauf liest der Interpreter einen Befehl des Programms ein, wertet ihn aus und führt danach die zugehörige Aktion durch. Die Ursache für die langsame Abarbeitung eines interpretierten Programms liegt in dieser Vielzahl von Schritten, die bis zur Ausführung eines Befehls notwendig sind. Dieses Merkmal ist für alle Interpreter, beispielsweise auch für den GW-Basic-Interpreter, bezeichnend.

Der maßgebliche Vorteil eines Interpreters liegt darin, daß ein Programm während der Entwicklung jederzeit direkt gestartet werden kann, während beim Compiler das Programm zuerst übersetzt werden muß. Vor allem bei der Fehlersuche sind diese Eigenschaften eines Interpreters für den professionellen Programmierer von Bedeutung.

Ein Compiler hingegen geht anders vor: Bei der Kompilierung eines Programms liest er, genau wie der Interpreter, die Befehle des Programms einzeln ein und wertet sie aus. Doch statt die dem Befehl entsprechende Aktion direkt auszuführen, generiert er einige Maschinensprachebefehle und schreibt diese in eine neue Datei. So entsteht ein reines Maschinenprogramm, dessen Wirkung genau der des ursprünglichen Quelltextes entspricht, aber sonst nichts mehr mit diesem gemeinsam hat. Bei der späteren Ausführung des Maschinenprogramms fallen das Lesen und Auswerten des Quelltextes völlig weg, was die höhere Geschwindigkeit im Vergleich zum Interpreter erklärt. Auch ist die Existenz des Compilers während des Ablaufs eines von ihm übersetzten Programms nicht notwendig, da der Compiler in jedes Programm eine sogenannte Laufzeitumgebung einbindet. So ist ein kompiliertes Programm nicht nur um ein Vielfaches schneller, sondern es steht ihm während seiner Laufzeit auch mehr Speicherplatz zur Verfügung. Folglich ist zur Entwicklung von Software ein Interpreter das geeignetere Werkzeug, während bei der Anwendung eines getesteten Programms eine kompilierte Version zu bevorzugen ist.

Im Falle des Batch-Sprachen-Interpreters des Betriebssystems handelt es sich um eine besonders langsame Interpretervariante. Der Kommandointerpreter lädt die auszuführende Stapeldatei während der Laufzeit nämlich nicht in den Hauptspeicher, sondern liest jeden neuen Befehl direkt vom Datenträger (was auf Eigenarten der Interpretation zurückzuführen ist). Dieses Verfahren beansprucht zwar keinen

Speicherplatz für das auszuführende Programm, bremst aber dessen Abarbeitung sehr stark. Die CPU muß nämlich immer auf die Ein- und Ausgaben von der Festplatte oder Diskette warten. Das hatte unter anderem die Folge, daß bis einschließlich der MS-DOS-Version 2.11 innerhalb von Stapeldateien der Aufruf weiterer Stapeldateien nicht als Unterprogramm realisierbar war. Die Kontrolle ging vollständig an die aufgerufene Stapeldatei über, ohne daß eine Rückkehr erlaubt war. Der Call-Befehl, der bei neueren MS-DOS-Versionen diese Lücke füllt, wird auch von Batch_It nur indirekt unterstützt.

Um einen Compiler und dessen Arbeitsweise zu verstehen, bedarf es noch einiger Erklärungen. Diese gewähren Ihnen einen groben Überblick über den Bau von Compilern; ein umfassender Einblick würde den Rahmen dieses Beitrags sprengen. Für Interessierte sei das Buch von Aho, Sethi und Uhlman empfohlen (siehe Literaturhinweis). Es ist ein Standardwerk zu diesem Thema und sollte im Bücherschrank eines jeden vorhanden sein, der den Entwurf eines Compilers plant.

Aufgabe eines Compilers ist es, ein im Quelltext vorliegendes Programm einer Sprache in eine andere Sprache zu übersetzen. Bei der Zielsprache muß es sich nicht unbedingt um den Maschinencode eines Prozessors handeln. Es kann beispielsweise auch eine Zwischensprache sein, die anschließend interpretiert (Pseudocode) oder weiterübersetzt wird. Damit erreicht man die bessere Portierbarkeit eines Compilers auf einen anderen Prozessor. Der Pseudocode ist so aufgebaut, daß eine Interpretation schneller vonstatten geht als beim ursprünglichen Quelltext. Meist handelt es sich dabei um reine Zahlenwerte, mit denen ein Prozessor bekanntermaßen am besten umgehen kann.

Die Übersetzung von Quelltexten gliedert sich im weitesten Sinne in folgende Teilaufgaben:

- Einlesen der Programmbefehle und -strukturen
- lexikalische Analyse
- syntaktische Analyse
- semantische Analyse
- Codeerzeugung (auch Zwischencode)

Diese Teilaufgaben werden traditionell auf mehrere Ebenen verteilt, die logisch aufeinander aufbauen, aber dennoch voneinander unabhängig sind (Bild 1). Da der Sprachumfang der Batch-Sprache sehr klein ist, ist diese prädestiniert, die Funktionsweise eines Compilers zu erläutern.

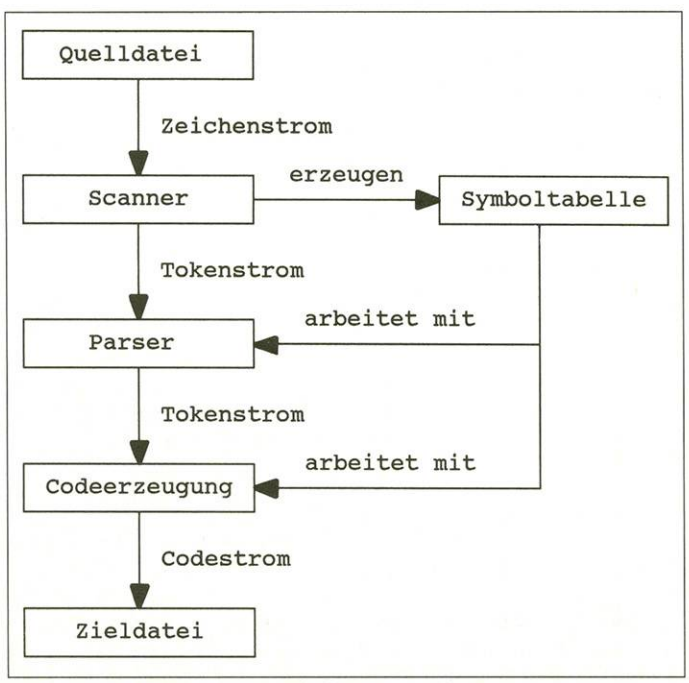


Bild 1. Die Ebenen eines einfachen Compilers

	Scannerinformation		Parserinformation	
Nr.	Lexem	Token	C_Adr	emit
0	\n	2	...	...
1	==	3	...	...
2	@	1	...	...
3	echo	100	...	...
4	call	101	...	...
5	for	102	...	...
637	edit	5	...	...

Bild 2. Der Aufbau der Symboltabelle von Batch_It

Mit Batch_It steht Ihnen ein einfacher Ein-Pass-Compiler zur Verfügung, der keinen Zwischencode, sondern sofort ausführbare Com-Dateien erzeugt. Er besteht aus dem Scanner »scan.c« (Listing 1), der den Quelltext einliest und gruppiert sowie einem Parser, der die vom Scanner erzeugten Symbole überprüft und den Code generiert. Diese Bauteile finden Sie in jedem Compiler, ihre Funktionsweise lernen Sie im folgenden näher kennen.

Der sogenannte Scanner bildet die erste Stufe eines Compilers. In seinen Aufgabenbereich fällt das Einlesen des Quelltextes, die Gruppierung der im Text enthaltenen Eingabezeichen und die Erzeugung von Symbolen, die der Parser weiterverarbeitet. Außerdem entfernt der Scanner die Kommentare und Leerzeichen aus dem Quelltext, da diese für die Übersetzung völlig unerheblich sind. Der Scanner verwaltet ebenso die Variablen für die Zeilen- und Spaltenposition des Lesezeigers und gibt bei Zeichen, die er nicht einordnen kann, erste Fehlermeldungen aus. Das dafür zuständige Modul des Compilers Batch_It heißt »error.c« (Listing 2).

Der Scanner erzeugt Tokens, die dem Parser die Arbeit erleichtern

Bei Batch_It ist der Scanner in zwei Phasen unterteilt. Die erste, durch die Funktion »nextchar« realisierte Phase, liest ein neues Zeichen des Quelltextes. »nextchar« liest den Quelltext Zeile für Zeile direkt vom Datenträger, hält den Text also nicht vollständig im Hauptspeicher. Damit ist immer nur die aktuell zu übersetzende Zeile im Eingabepuffer »linebuf«. Der Compiler übersetzt daher nahezu beliebig große Quelltexte, was bei der Alternative, den gesamten Quelltext vor der Übersetzung in den Speicher zu laden, nicht der Fall ist.

Die Funktion »gettoken« realisiert die zweite Phase, die aus den von »nextchar« gelieferten Zeichen Symbole (meist bestimmte Zahlenwerte für bestimmte Befehle) generiert und zurückliefert. Die im Sprachumfang enthaltenen Kommandos nennt man Schlüsselwörter. Jedem Schlüsselwort ordnet man ein ganz bestimmtes (eindeutiges) Symbol zu. Die Symbole (auch Token genannt) vereinfachen die weitere Verarbeitung und gestalten diese übersichtlicher.

Ein Parser muß sich also weder um Kommentare noch um Leerzeichen kümmern. Auch bei der Verarbeitung von Befehlen arbeitet er mit einem Token, statt mit den Zeichenketten der Kommandos, deren Länge unterschiedlich ist. Zur

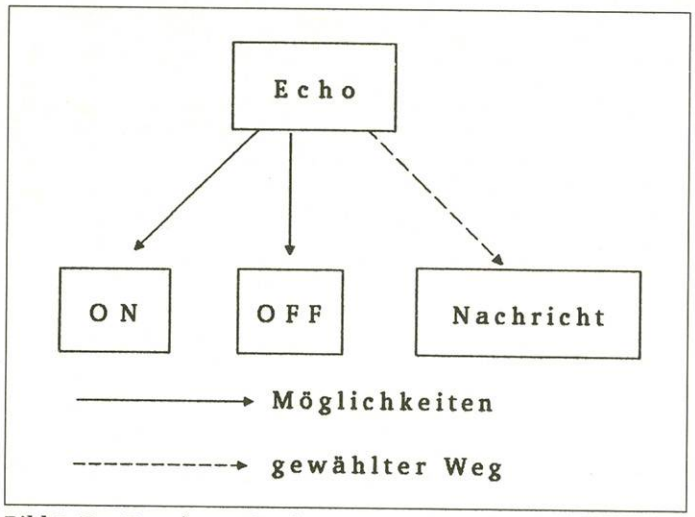


Bild 3. Der Parsebaum für das Kommando »echo«

Erkennung, Verwaltung und Zuordnung der Zeichenketten und Symbole dient die sogenannte Symboltabelle, in der die Lexeme, das heißt, die zu einem Symbol gehörende Zeichenkette, aufbewahrt werden. In der Symboltabelle sind zu jedem Eintrag noch einige andere Informationen gespeichert, die aber nur der Parser verwendet (Bild 2). Die Funktionen, die der Scanner für die Verwaltung der Symboltabelle benötigt, finden Sie in der Datei »dict.c« (Listing 3). Die Bedeutungen der drei wichtigsten Funktionen wird hier kurz erläutert:

- »lookup« überprüft, ob ein Eintrag schon vorhanden ist und gibt dessen Schlüsselnummer zurück.
- »enterstring« generiert einen neuen Eintrag in der Symboltabelle, falls kein Eintrag mit der Zeichenkette »string« existiert.
- »enterkeys« trägt die Kommandos der Batch-Sprache (Tabelle), die sogenannten reservierten Schlüsselwörter, in die Symboltabelle ein und ordnet ihnen ihre Symbole zu, die der Parser zur Verarbeitung benötigt.

Die Überprüfung fehlerhaft geschriebener Kommandos findet beim Scanner von Batch_It nur insoweit statt, als daß nicht erkannte Befehle als externe Kommandos in die Symboltabelle eingetragen werden. Da es in einer Stapeldatei gestattet ist, beliebige Anwendungen mit beliebigen Namen aufzurufen, sind falsch geschriebene Kommandos nicht zu erkennen, und der Fehler macht sich oft erst während der Abarbeitung bemerkbar.

Nachdem der Scanner den eigentlichen Quelltext in Symbole zerlegt und von überfüssigen Leerzeichen und Kommentaren befreit hat, ist der Parser mit der Bearbeitung an der Reihe. Der Aufgabenbereich eines Parsers besteht im Aufbau von sogenannten Parsebäumen, mit denen die Richtigkeit der im Quelltext verwendeten Syntax überprüft wird. Im Falle von Batch_It erzeugt der Parser auch den ausführbaren Code. Diese Verbindung von Syntaxkontrolle und Codeerzeugung ist bei sogenannten Top-Down-Parsern durchaus üblich. Um den Sinn und die Funktion von Parsebäumen zu verstehen, folgen jetzt einige Begriffsdefinitionen.

Jede existierende Sprache besitzt Regeln für ihren Gebrauch. Diese Sprachregeln nennt man in der Linguistik Grammatik. Sie bestimmen, wie Sätze in einer Sprache zu bilden sind. Auch für Programmiersprachen existieren solche Grammatiken, die üblicherweise in sogenannten Produktionen dargestellt werden. Als Beispiel betrachten Sie das Batch-Kommando »echo«. Auf das Kommando »echo« folgt entweder »on«, »off« oder eine auszugebende Nachricht, die bis zum Zeilenende reicht und aus einer beliebigen Zeichenkette besteht. Als Produktion geschrieben sieht dieses Kommando folgendermaßen aus:

```
echo -> on ¦ off ¦ Nachricht
Nachricht -> Zeichen ¦ Newline
Zeichen -> Zeichen ¦ ϵ
```

Die verwendeten grafischen Symbole haben dabei folgende Bedeutung:

```
N -> T : N produziert T
L ¦ R : L oder R
ϵ : leerer String
```

Alle Kommandos einer Sprache sind in dieser Schreibweise darstellbar. Dabei sind einige Sonderfälle zu beachten, auf die hier aus Platzgründen nicht näher eingegangen werden kann. Liefert der Scanner das Symbol »echo« an den Parser, durchläuft dieser den sogenannten Parsebaum für dieses Symbol, der als ein Weg durch die Produktionen zu deuten ist. Als Beispiel nehmen Sie an, daß im Quelltext folgende Kommandozeile steht:

```
echo Hallo Welt
```

Im als Parsebaum dargestellten Format der Produktionen wird ein bestimmter Ast des Baumes durchlaufen und so die Zugehörigkeit der Kommandozeile zur Batch-Sprache festgestellt (Bild 3).

Mit Parsebäumen überprüfen Sie die Syntax der Eingabe auf Richtigkeit

Allgemein kann es bei Parsebäumen und Produktionen zu verschiedenen Arten von Mehrdeutigkeiten und Rekursionen kommen. Es ist zwar nicht ganz einfach, diese aufzulösen, die Automatentheorie liefert jedoch die notwendigen Algorithmen dazu. Wichtig ist an dieser Stelle, daß Sie die Funktionsweise eines Parsers durchschauen, deshalb ein Beispiel für eine Produktion und einen Parsebaum, wie er auch in Hochsprachen-Compilern vorkommt. Es handelt sich dabei um eine Produktion zur Überprüfung von numerischen Ausdrücken wie beispielsweise »5 + 2 – 1«.

Diese Produktion überprüft also, ob die eingelesenen Zeichen einen numerischen Ausdruck darstellen oder nicht. Erlaubt sind in diesem Beispiel nur einstellige Ziffern, die durch die Operatoren »+« und »–« verbunden sind. Die Produktionen dazu lauten:

```
Ausdruck -> Ziffer + Ausdruck ¦
            Ziffer – Ausdruck ¦
            Ziffer ϵ
            Ziffer -> 0 ¦ 1 ¦ 2 ¦ 3 ¦ 4 ¦
                      5 ¦ 6 ¦ 7 ¦ 8 ¦ 9
```

Den zu dem Ausdruck »5 + 2 – 1« gehörenden Parsebaum finden Sie in Bild 4.

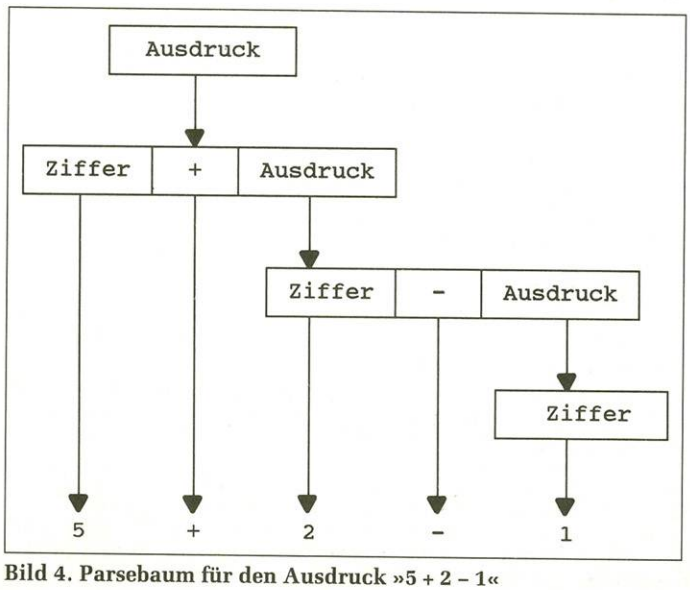


Bild 4. Parsebaum für den Ausdruck »5 + 2 – 1«

Für die softwaremäßige Realisierung von Parsern existieren mehrere Alternativen. Man unterscheidet hierbei die Top-Down-Parser von den Bottom-Up-Parsern, die beide versuchen, einen Parsebaum für einen eingelesenen Ausdruck aufzubauen und so die Korrektheit der verwendeten Syntax zu überprüfen. Beide Methoden liefern prinzipiell dasselbe Ergebnis, werden aber bei unterschiedlichen Gelegenheiten eingesetzt. Im folgenden lernen Sie die Arbeitsweise eines speziellen Top-Down-Parsers kennen, der nach der sogenannten Recursive-Descent-Analyse vorgeht: den prädiktiven Parser. Bei diesem Verfahren setzen Sie einfach jede Produktion in eine Prozedur um. Die Prozeduren rufen dabei sich selbst und andere Prozeduren auf und erzeugen so einen Parsebaum. Für die oben dargestellten Produktionen zur Überprüfung von numerischen Ausdrücken ergibt sich daher der in Bild 5 gezeigte Listingauszug.

Da ein solcher Parser nicht nur einen Parsebaum erzeugt, sondern da auch in jeder Phase eindeutig ist, mit welchen Kommando- und Typenüberprüfungen er sich gerade beschäftigt, bietet es sich geradezu an, in diese Parseprozeduren auch die Codeerzeugung miteinzubeziehen. Auf diese Weise wird ein weiterer Codedurchlauf und die Erzeugung einer Datei für den vom Parser erzeugten Zwischencode eingespart. Auch bei Batch_It erfolgt die Codegenerierung während des Parsings. Im Normalfall erzeugt ein Compiler an dieser Stelle nicht das ausführbare Programm, sondern einen Zwischencode. Dieses Vorgehen hat mehrere Vorteile. Mit der Trennung eines Compilers in Front-End und Back-End halten Sie dessen Front-End maschinenunabhängig. Das erleichtert eine schnelle Portierung des Compilers auf andere Rechner, bei der lediglich ein neues Back-End geschrieben werden muß. Des weiteren kann eine Optimierung dieses Zwischencodes die Effizienz des ausführbaren Codes erhöhen.

Noch ein paar Bemerkungen zu Front-End und Back-End. Das Front-End besteht aus den Phasen, die in erster Linie von der Quellsprache abhängig und von der Zielmaschine unabhängig sind. Dazu zählen die lexikalische und semantische Analyse, der Aufbau der Symboltabelle und die Generierung des Zwischencodes. Schon in dieser Phase kann eine Optimierung des Zwischencodes erfolgen. Die Teile des Compilers, die auf die Zielmaschine zugeschnitten sind, sind im Back-End enthalten. Logischerweise sind diese Teile lediglich von der Zwischensprache und nicht von der Quellsprache abhängig. Hier wird der Zielcode erzeugt und eventuell optimiert.

Bei Batch_It wurde auf die Generierung sowohl von Zwischencode als auch von Assembler-Quelltext verzichtet, da einerseits eine Portierung wohl auszuschließen ist und andererseits nicht jeder Leser mit dem MASM oder TASM ausgestattet ist.

```
void Ziffer()
{ if(isdigit(nexttoken())  return
  else error; /* keine Ziffer => Fehler */
}

void Ausdruck()
{ Ziffer();
  switch(nexttoken())
  { case '+':
    case '-':
      Ausdruck();
    break;
   default:
     retract(); /* Ende des Ausdrucks */
     /* Lesezeiger eins zurück */
     break;
  }
}
```

Bild 5. Der Listingauszug zur Überprüfung eines numerischen Ausdrucks

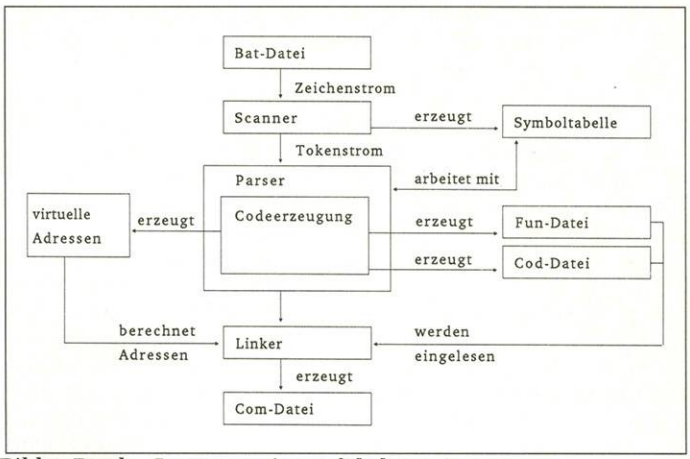


Bild 6. Batch_It erzeugt ein ausführbares Programm

Batch_It ist ein Ein-Pass-Compiler, das bedeutet, er liest den Quelltext ein einziges Mal, um ihn zu kompilieren. Für jede Quelltextdatei, die Batch_It kompiliert, erzeugt er zwei Arbeitsdateien mit den Endungen »fun« und »cod«. Dabei werden in der Datei mit der Endung »fun« die Funktionen, Prozeduren, Variablen und Daten gespeichert und in der Cod-Datei das eigentlich Hauptprogramm, das die Funktionen, Prozeduren und Daten der Fun-Datei benutzt. Die Fun-Datei wird im weiteren als Datenbereich, die Cod-Datei als Codebereich bezeichnet. Die beiden Dateien werden nach der eigentlichen Übersetzung von der Prozedur »link« aus »batch_it.c« (Listing 4) zur ausführbaren Com-Datei zusammengebunden. In beiden Arbeitsdateien arbeitet Batch_It mit virtuellen Adressen, die erst beim Zusammenbinden von »link« auf ihren eigentlichen Wert gesetzt werden. Den Ablauf einer Kompilierung mit Batch_It sehen Sie in Bild 6. Die Datei »asmdef.h« (Listing 5) enthält die für die Codegenerierung benötigten Prozessorbefehle als Konstanten. Die Routine »asm_it« verwendet diese Konstanten, um den Code zu erzeugen. Allerdings ist »asm_it« nur ein »#define«, das die Routine »arg_exe« aus dem Modul »arg_do.c« (Listing 6) verwendet.

Jedes Batch-Programm verlegt die Disk Transfer Area (DTA)

Das schließlich erzeugte ausführbare Programm liegt in einem Format vor, das aus mehreren verschiedenen Segmenten besteht. Das Programm beginnt mit einem Sprungbefehl, der zum eigentlichen Hauptprogramm führt. Direkt hinter dem Sprungbefehl folgt der Bereich der neuen Disk Transfer Area (DTA). Beim Start eines Programms unter MS-DOS liegt die DTA im sogenannten Program Segment Prefix (PSP), einem Variablenblock, der jedem Programm im Hauptspeicher voransteht. Da aber im gleichen Speicherbereich wie die DTA auch die Argumente der Kommandozeile gespeichert sind, auf die während des Programmablaufs zugegriffen wird, verlegt jedes von Batch_It erzeugte Programm zuerst einmal die DTA. Wenn ein Kommando oder eine Anwendung die neue DTA benutzt, überschreibt es damit nicht sofort die übergebenen Argumente. Darauf folgt der Bereich, in dem die Daten, Funktionen und Prozeduren liegen: der Datenbereich. Die Prozeduren und Funktionen werden je nach Bedarf von Batch_It eingebunden; der Datenbereich hat deshalb kein festes Aussehen. Hinter dem Datenbereich folgt der Initialisierungsteil des Hauptprogramms, in dem die Verlegung der DTA und die Freigabe des restlichen Hauptspeichers stattfindet. Nun folgt der eigentliche Codebereich, der mit den Informationen aus

dem Datenbereich arbeitet und auch die Funktionen aufruft. Mit der Beendingung des Programms endet auch die Com-Datei.

Die Com-Datei ist, wie schon beschrieben, in Code- und Datenbereich unterteilt. Der erste Befehl im Datenbereich ist ein »jmp« zur Initialisierungsroutine, dann folgen die neue DTA, die Routinen und Daten. Der Codebereich setzt sich aus der Initialisierung, den Funktionsaufrufen und der Beendigungsroutine zusammen.

Batch_It setzt nicht alle übersetzten Kommandos in eine Folge von Maschinensprachebefehlen um. Der Compiler erzeugt auch Aufrufe auf Funktionen und Prozeduren im Datenbereich, die äquivalente Aktionen zu den jeweiligen Batch-Kommandos darstellen. So wird eine Funktion, die für ein Kommando notwendig ist, nur einmal im Datenbereich gespeichert und dann bei Bedarf im Codebereich aufgerufen. Dies macht sich vor allem bei längeren Stapeldateien bemerkbar, deren Com-Datei pro neuem zusätzlichem Kommando kaum noch an Länge zunimmt. Durch das ausschließliche Hinzufügen von benötigten Funktionen und Prozeduren enthält jede übersetzte Stapeldatei eine für sie spezifische Bibliothek, die keinen unnötigen Speicherplatz verschwendet.

Die beiden wichtigsten internen Prozeduren kompilierter Stapeldateien sind »interpret« und »dummy«. »interpret«, die längste Prozedur, wird automatisch eingebunden, wenn eine Stringverarbeitung stattfindet. Sie ersetzt die Batch-Variablen »%1« bis »%9« durch die Argumente, die Sie in der Kommandozeile des Aufrufs übergeben haben. So wird beispielsweise aus der Zeile »echo %1« in einer Stapeldatei »test.bat«, die mit »test.bat Hello« aufgerufen wurde, schließlich »echo Hello«. »interpret« nimmt diese Ersetzungen für die Kompilate vor und besteht aus vier Teilen: der Hauptschleife und drei Unterprogrammen. In der Hauptschleife durchsucht »interpret« die Zeichenkette nach Batch-Variablen. Findet die Funktion eine solche Variable, so stellt Unterprogramm 1 fest, wo und wie lange der in der Kommandozeile übergebene Parameter ist. Unterprogramm 2 schafft Platz für den übergebenen Parameter und kopiert die durchsuchte Zeichenkette um dessen Länge nach rechts. Schließlich kopiert Unterprogramm 3 den gefundenen Parameter an die richtige Stelle in der durchsuchten Zeichenkette. Unterstützt wird dabei natürlich auch das Shift-Kommando, mit dem Sie auf mehr als zehn übergebene Parameter zugreifen können.

Da Sie den Sprachumfang der Batch-Sprache erweitern und somit beliebige Anwendungen aus einer laufenden Stapeldatei starten, muß Batch_It auch noch nicht bekannte Befehle ausführen können. Dazu gibt es mehrere konventionelle Alternativen, die jedoch Nachteile aufweisen.

Beispielsweise können Sie unter MS-DOS über die Exec-Funktion 4Bhex des Interrupts 21hex Programme einladen und starten. Batch_It könnte nach dieser Strategie vorgehen und alle unbekannten Befehle über diesen Interrupt aufrufen. Dies ließe sich auch für normale Anwendungsprogramme und externe Befehle von MS-DOS realisieren. Bei den internen Befehlen entsteht jedoch beim Einsatz dieser Methode folgendes Problem: Die internen Kommandos von MS-DOS werden nur vom Kommandointerpreter selbst verstanden und umgesetzt. Mit dem Interrupt 4Bhex lassen sich keine internen Kommandos starten.

Eine Lösung für dieses Dilemma ist, einen neuen Kommandointerpreter über die Funktion 4Bhex zu starten und diesem den Parameter »/c«, gefolgt vom zu startenden Programm, zu übergeben. Folgende Zeichenkette ist beispielsweise der Funktion 4Bhex zu übergeben, um den Dir-Befehl auszuführen:

```
command /c dir
```

Es gibt leider eine Eigenheit der internen Befehle, die auch dieses Verfahren disqualifiziert. Falls ein Befehl einer Stapeldatei die Umgebungsvariablen des ursprünglichen »command.com« verändert und dieser Befehl über den neuen Kommandointerpreter gestartet wird, verändert der Befehl nur die Umgebungsvariablen des neuen Kommandointerpreters. Die Umgebungsvariablen des alten »command.com« bleiben bestehen.

Stellen Sie sich vor, Batch_It stößt bei der Kompilierung einer Stapeldatei auf das Kommando »prompt«, das nicht in seinem Sprachumfang enthalten ist. Falls Sie dieses Kommando über die Exec-Funktion starten, passiert nach außen hin weder mit noch ohne Laden eines »command.com« etwas. Zwar führt ein neu gestarteter »command.com« den Befehl aus, doch wirkt der Befehl lediglich auf die Umgebungsvariablen des neuen Kommandointerpreters, nicht auf die des Interpreters, der die Stapeldatei abarbeitet.

Ein weitere Alternative wäre die selbständige Verwaltung der Umgebungsvariablen des betroffenen »command.com«. Dazu ist eine Funktionsbibliothek notwendig, die alle internen Kommandos zur Veränderung von Umgebungsvariablen emuliert. Der Compiler müßte diese Kommandos abfangen und sie durch Aufrufe aus dieser Bibliothek ersetzen. Dieser Weg ist zwar realisierbar, jedoch mit hohem Entwicklungsaufwand für die Funktionsbibiothek verbunden.

Programmaufruf über eine undokumentierte MS-DOS-Funktion

Bei Batch_It wurde ein drittes Verfahren gewählt. Alle Kommandos, die der Compiler nicht in seinem Befehlssatz findet, startet er über die sogenannte »backdoor« von MS-DOS, den Interrupt 2Ehex. Über ihn rufen Sie Kommandos und Anwendungen so auf, als würden Sie diese in der Kommandozeile des ursprünglichen »command.com« eintippen und die Eingabe-Taste drücken. Dabei wird der schon im Speicher befindliche Kommandointerpreter angesprochen. Das bedeutet auch, daß beim Aufruf von Kommandos, die dessen Umgebungsvariablen verändern, diese auch wirklich neu gesetzt werden. Mit dieser undokumentierten Funktion von MS-DOS vereinen sich die Vorteile der oben genannten Verfahren, während deren Nachteile entfallen: Zum einen ist es nicht notwendig, eine Funktionsbibliothek zu entwickeln, zum anderen wächst die Funktion 2Ehex mit den neuen MS-DOS-Versionen mit, da bei ihrem Aufruf immer der originale Kommandointerpreter benutzt wird. Zudem entfällt der Geschwindigkeitsverlust, den das Laden eines neuen »command.com« mit sich bringt.

Sowohl das Kommando »echo« als auch die für Stapeldateien übliche Ausgabe der gerade bearbeiteten Kommandozeile werden über die MS-DOS-Funktionen abgewickelt. Die Verwendung von Labels wie »:Marke« beeinflußt den später erzeugten Code in keiner Weise, da alle Sprungadressen noch während des Kompiliervorgangs ausgewertet und

Kommando	Bedeutung
@	bewirkt, daß die Kommandos dieser Zeile nicht angezeigt werden
:Label	definiert eine Marke für die Anweisung »goto«
call	ruft eine andere Stapeldatei auf
echo	schaltet die Anzeige der Befehlszeile ein oder aus und gibt Nachrichten aus
for	wiederholt die Ausführung eines Kommandos
goto	springt zur angegebenen Marke
if	bedingte Verzweigung
pause	wartet auf einen Tastendruck
rem	zeigt an, daß der folgende Text ein Kommentar ist
shift	verschiebt die übergebenen Parameter der Kommandozeile

Die Kommandos der Batch-Sprache

beim Zusammenbinden berechnet werden. So sind alle Goto-Anweisungen und selbst Sprünge, die durch das If-Kommando notwendig werden, als direkte Near-Jumps in Maschinensprache umgesetzt.

Den Call-Aufruf, der es gestattet, aus einer Stapeldatei eine andere zu starten, übersetzt Batch_It bisher nicht. Trotzdem können Sie ihn verwenden. Da Batch_It ihn wie ein unbekanntes Kommando behandelt, läuft die bezeichnete Stapeldatei wirklich als Stapeldatei ab.

Der Compiler Batch_It beinhaltet keinen Bildschirmeditor, wie er Ihnen aus dem Lieferumfang von »großen« Compilern wie Turbo Pascal bekannt sein dürfte. Sie starten ihn daher von der Kommandozeile aus.

Um beispielweise das bekannte Beispielprogramm zu erzeugen, das die Zeichenkette »Hello, world« auf dem Bildschirm ausgibt, gehen Sie folgendermaßen vor: Schreiben Sie eine Datei mit dem Namen »hello.bat« und dem Inhalt »@echo Hello, world«. Nach dem Sichern dieser Datei geben Sie

```
batch_it hello.bat
```

ein. Nach der kurzen Übersetzungszeit befindet sich im selben Verzeichnis die Datei »hello.com«. Dies ist ein vollständiges und unabhängiges Maschinenprogramm, das bei seinem Aufruf genau dieselbe Aktion ausführt, wie die entsprechenden Hochsprachenbeispiele. Mit dem winzigen Unterschied, daß das Programm mit Turbo C kompiliert 9446 Byte, mit Quick C 7494 Byte und mit Batch_It gerade 413 Byte Länge einnimmt.

Außer den Geschwindigkeitsvorteilen, die ein durch Batch_It übersetztes Programm erfährt, existieren noch einige andere kleine Unterschiede. Wie oben bereits erläutert, benutzt Batch_It den undokumentierten Interrupt 2Ehex zum Ausführen unbekannter Kommandos. Bei einigen nicht ganz perfekten Klonen des Betriebssystems MS-DOS ist an diesen Interrupt wohl nicht gedacht worden, obwohl er sowohl bei MS-DOS als auch bei DR-DOS eine nicht zu unterschätzende Bedeutung einnimmt. Beispielsweise ist bei 4Dos, einem Tool, das den Kommandointerpreter ersetzt, dieser Interrupt nicht vorhanden. Dies führt dazu, daß die kompilierten Programme unter 4Dos nicht korrekt laufen.

Auch unter DR-DOS ist der Batch-Compiler nicht ohne Einschränkungen einzusetzen. Dort funktionieren die Befehle nicht, die auf die Umgebungsvariablen wirken.

Außerdem sollten Sie keine Stapeldateien übersetzen, die TSR-Programme installieren, wie beispielsweise Tastaturtreiber oder ähnliches. Bei der Abarbeitung einer echten Stapeldatei befindet sich diese nicht im Speicher, sondern wird, wie erwähnt, befehlsweise von Datenträgern eingelesen. Beim Aufruf von TSR-Programmen werden diese korrekt in den ersten freien Hauptspeicherbereich gelegt. Da aber eine von Batch_It übersetzte Stapeldatei bei ihrem Ablauf als Programm geladen und ausgeführt wird, liegen die TSR-Programme oberhalb des ablaufenden Programms. Ist dann die kompilierte Stapeldatei abgelaufen, wird der von ihr belegte Speicherbereich wieder freigegeben. Damit liegt das installierte TSR-Programm mitten im Hauptspeicher und verhindert, daß größere Anwendungen, die einen kontinuierlich großen Speicherblock zum Ablauf benötigen, abgearbeitet werden können.

Wenn Sie diese Eigenheiten beachten, wird Ihnen Batch_It ein nützlicher Helfer sein.

(Matthias Oberdorfer/rm)

Literatur:

Compilerbau Teil 1: Aho, Sethi, Uhlman, Addison-Wesley, 576 Seiten, ISBN 3-89319-150-X

Compilerbau Teil 2: Aho, Sethi, Uhlman, Addison-Wesley, 371 Seiten, ISBN 3-89319-151-8

Undocumented DOS, Andrew Schulman, Addison-Wesley, 694 Seiten, ISBN 0-201-57064-5

```
[UD2]    1: /*Datei: scan.c
[AC5]    2:   Funktion: liefert Token zurück
[1R2]    3:   Sprache: Turbo C
[3L4]    4:   Autor: Matthias Oberdorfer
[T15]    5:   (c)1991 DMV Widuch GmbH & Co. KG */
[ØØØ]    6:
[8W2]    7: #include<stdio.h>
[XU2]    8: #include<ctype.h>
[ØØØ]    9:
[F43]   1Ø: #define MAXFILES 1Ø
[ØØØ]   11:
[8M2]   12: typedef struct
[5T2]   13: { FILE *f_handle;
[RR2]   14:   unsigned zeile;
[X72]   15:   unsigned key;
[EWØ]   16: }BAT;
[ØØØ]   17:
[M73]   18: BAT bat[MAXFILES];
[Ø13]   19: unsigned act_f=-1;
[562]   2Ø: int this_ch;
[ØØØ]   21:
[Ø43]   22: #define MAXBUF Øx3ØØ
[ØØØ]   23:
[2T6]   24: char linebuf[MAXBUF],linebuf2[MAXBUF];
[CE3]   25: unsigned linesize=Ø;
[544]   26: int linepos=1,tokpos=1;
[ØØØ]   27:
[ZK2]   28: #include"dict.c"
[ØØØ]   29:
[AC2]   3Ø: int nextchar()
[V66]   31: { if(linepos>linesize || !linesize)
[Z65]   32:   { linesize = linepos = tokpos=Ø;
[LZ1]   33:     /* read new */
[FM1]   34:     do /*line */
[1D5]   35:     { linebuf[linesize++]=this_ch=
[LK4]   36:         getc(bat[act_f].f_handle);
[HR3]   37:     }while(this_ch !='\n' &&
[6M2]   38:            this_ch != '\r' &&
[YE2]   39:            this_ch != EOF &&
[K23]   4Ø:            linesize<MAXBUF &&
[CF4]   41:            this_ch != Øx1a); /* Strg-Z */
[5W3]   42:     linebuf[--linesize]=Ø;
[6A5]   43:     if((this_ch==EOF && !linesize)||
[4B4]   44:       (this_ch==Øx1a&& !linesize))
[C72]   45:     { linesize=Ø;
[UY1]   46:       return(EOF);
[U7Ø]   47:     }
[E16]   48:     this_ch=getc(bat[act_f].f_handle);
[H75]   49:     if(this_ch!='\n' && this_ch!='\r')
[Z76]   5Ø:     ungetc(this_ch,bat[act_f].f_handle);
[Y93]   51:     bat[act_f].zeile++;
[U7Ø]   52:   }
[Y14]   53:   if(linepos<Ø)  /* NEWLINE */
[RY1]   54:   { linepos++;
[6G3]   55:     return(this_ch='\n');
[U7Ø]   56:   }
[8L3]   57:   if(linepos==linesize)
[RY1]   58:   { linepos++;
[6G3]   59:     return(this_ch='\n');
[U7Ø]   6Ø:   }
[366]   61:   return(this_ch=linebuf[linepos++]);
[U7Ø]   62: }
[ØØØ]   63:
[5V5]   64: #define retract this_ch=linebuf[\
[TZ1]   65:         --linepos -1]
[ØØØ]   66:
[WA2]   67: open_bat(name)
[ØU1]   68: char *name;
[LZ3]   69: { bat[++ act_f].zeile=Ø;
[Y24]   7Ø:   if((bat[act_f].f_handle=
[LS3]   71:       fopen(name,"rb"))==NULL)
[H72]   72:     return(NULL);
[UL1]   73:   return(1);
[U7Ø]   74: }
[DX1]   75: close_bat()
[FA5]   76: { fclose(bat[act_f--].f_handle);
[U7Ø]   77: }
[ØØØ]   78:
[4B6]   79: /* int gettoken() :liefert den Schlüssel
[EV4]   8Ø:    auf die keytabelle zurück */
[ØØØ]   81:
[A33]   82: int gettoken(flag)
[9R1]   83: word flag;
[R83]   84: { char lexbuf[Øx1ØØ];
[XD3]   85:   unsigned bufsize=Ø;
[BU2]   86:   tokpos=linepos;
[NBØ]   87:   do
[W93]   88:   { switch(nextchar())
[6W5]   89:     { case ' ': /* Leerzeichen übergehen */
[PØ1]   9Ø:       break;
[6U4]   91:       case EOF: /* EOF zuückgeben */
[AZ2]   92:         return(EOF_SCAN);
[US5]   93:       case '\n': /* NEWLINE zurückgeben */
[DK1]   94:       case '\r':
[NT4]   95:         lexbuf[Ø]='\n'; lexbuf[1]=Ø;
[SG4]   96:         return(enterstring(lexbuf,
[FP1]   97:                            NEWLINE));
[Y81]   98:       case '=':
[7R3]   99:         lexbuf[bufsize++]='=';
```

Listing 1. »scan.c« liefert die Tokens für den Parserdurchlauf

```
[3G4]  1ØØ:          if(nextchar()!='=') retract ;
[ZRØ]  1Ø1:          else
[7R3]  1Ø2:            lexbuf[bufsize++]='=';
[9C3]  1Ø3:          lexbuf[bufsize]=Ø;
[XA5]  1Ø4:          return(enterstring(lexbuf,OP));
[A91]  1Ø5:        case '@':
[K81]  1Ø6:        case ':':
[4H5]  1Ø7:          lexbuf[Ø]=this_ch; lexbuf[1]=Ø;
[4Ø5]  1Ø8:          return(enterstring(lexbuf,Ø));
[8E1]  1Ø9:        default:
[NBØ]  11Ø:        do
[Ø45]  111:        { lexbuf[bufsize++]=this_ch;
[LY2]  112:          if(this_ch==' ' ||
[Ø33]  113:             this_ch == '\n' ||
[4D3]  114:             this_ch == '\r'  ||
[525]  115:             this_ch == EOF || this_ch =='=')
[PØ1]  116:           break;
[AW1]  117:          nextchar();
[3N1]  118:        }while(1);
[KL3]  119:        lexbuf[--bufsize]=Ø;
[UG5]  12Ø:        if(flag)strcpy(linebuf2,linebuf);
[3N2]  121:        strupr(lexbuf);
[BF1]  122:        retract;
[YV1]  123:        if(bufsize)
[GV5]  124:          return(enterstring(lexbuf,IDENT));
[U7Ø]  125:      }
[3N1]  126:    }while(1);
[U7Ø]  127: }
```

Listing 1. »scan.c« liefert die Tokens für den Parserdurchlauf

```
[UH2]    1: /*Datei: error.c
[UT5]    2:   Funktion: gibt Fehlermeldungen aus
[1R2]    3:   Sprache: Turbo C
[3L4]    4:   Autor: Matthias Oberdorfer
[T15]    5:   (c)1991 DMV Widuch GmbH & Co. KG */
[ØØØ]    6:
[MK1]    7: /* Fehler */
[NL3]    8: #define UNKNOWN          Ø
[2L3]    9:  /* Unbekannter Fehler */
[CU3]   1Ø: #define NO_MEMORY       -1
[KG4]   11:  /* Kein Speicherplatz mehr */
[8L4]   12: #define ENTRY_NOT_FOUND -2
[Y64]   13:  /* Eintrag nicht gefunden */
[S34]   14: #define KEYTABLE_OVL    -3
[FØ5]   15:  /* Schlüsseltabelle zu klein */
[CP3]   16: #define OPEN_FILE       -4
[974]   17:  /* File nicht zu öffnen */
[AM3]   18: #define NO_FILE         -5
[SK4]   19:  /* Kein Datei-Name angegben */
[8T3]   2Ø: #define EOF_SCAN        -1ØØ
[2D4]   21:  /* Ende der Datei erreicht */
[ØØØ]   22:
[5P2]   23: char *errors[]=
[5R4]   24: { "unknow error","no memory",
[366]   25:   "entry not found","keytable overflow",
[FG5]   26:   "can't open file","no file given" };
[ØØØ]   27:
[L33]   28: int error(nr,info)
[AB1]   29: int nr;
[9U1]   3Ø: char *info;
[R25]   31: { char help=linebuf[linepos];
[7V4]   32:   char help2=linebuf[tokpos];
[395]   33:   printf("\nERROR %d : %s %s\n",-nr,
[XG3]   34:          errors[-nr], info);
[TK3]   35:   if(bat[act_f].zeile)
[YR3]   36:   { linebuf[linepos]=Ø;
[GA3]   37:     linebuf[tokpos]=Ø;
[876]   38:     printf("line %d : ", bat[act_f].zeile);
[NG3]   39:     printf("%s ", linebuf);
[L14]   4Ø:     linebuf[tokpos]=help2;
[US5]   41:     printf("¤%s³", &linebuf[tokpos]);
[FØ4]   42:     linebuf[linepos]=help;
[FT5]   43:     printf(" %s\n", &linebuf[linepos]);
[ZW1]   44:     return(nr);
[U7Ø]   45:   }
[U7Ø]   46: }
```

Listing 2. »error.c« gibt die Fehlermeldungen des Compilers während der Übersetzung aus

```
[LD2]    1: /*Datei: dict.c
[CM5]    2:   Funktion: generiert Wörterbuch
[1R2]    3:   Sprache: Turbo C
[3L4]    4:   Autor: Matthias Oberdorfer
[T15]    5:   (c)1991 DMV Widuch GmbH & Co. KG */
[ØØØ]    6:
[2T2]    7: #include<alloc.h>
[B53]    8: #include<string.h>
[8W2]    9: #include<stdio.h>
[MT2]   1Ø: #include<fcntl.h>
[8S2]   11: #include"error.c"
[CY3]   12: typedef unsigned word;
[VE3]   13: #define MAXENTRY  1ØØØ
```

```
[ØØØ]   14:
[8M2]   15: typedef struct
[392]   16: { char *lexem;
[EF4]   17:   /* Zeichenkette mit Lexem */
[9W3]   18:   word  token;     /* Token */
[3Ø2]   19:   word  c_adr;
[1A3]   2Ø:   /* Adresse für CALL */
[X72]   21:   void  (*emit)();
[EP6]   22:   /* Funktion, die Assembler-Code ausgibt*/
[UW6]   23: } entry; /* Eintrag in Schüsseltabelle */
[ØØØ]   24:
[FS5]   25: #define TOKEN(c) keytable[c].token
[XP5]   26: #define LEXEM(c) keytable[c].lexem
[9S5]   27: #define C_ADR(c) keytable[c].c_adr
[3C5]   28: #define EMIT(c)  keytable[c].emit
[ØØØ]   29:
[DP4]   3Ø: /* Unser Schlüsseltabelle */
[EL4]   31: entry keytable[MAXENTRY];
[WF6]   32: /* Zeigt auf nächsten freien Eintrag */
[1V2]   33: int lastentry=Ø;
[DT2]   34: unsigned exec =Ø;
[WØ2]   35: void dummy()
[4L2]   36: { unsigned adr;
[2A4]   37:   if(!exec) /* einmal reicht */
[6M5]   38:   { handle=f_handle; /* In FUNC-OBJ */
[H14]   39:     adr = f_count+FUNC_SEG;
[3R2]   4Ø:     asm_it(ØØ,ØØ,ØØ,ØØ);
[CE4]   41:     exec=f_count; /* Adresse */
[646]   42:     asm_it(CS,MOV_MS,SS,adr&Øxff,adr>>8,
[934]   43:            CS,MOV_MR,SP,(adr+2)&Øxff,
[724]   44:            (adr+2)>>8,INT,Øx2e,CLI,CS,
[WG4]   45:            MOV_SM,SS,adr&Øxff,adr>>8,CS,
[RG3]   46:            MOV_RM,SP,(adr+2)&Øxff,
[6S4]   47:            (adr+2)>>8,STI,PUSH_CS,POP_DS,
[Z44]   48:            PUSH_CS,POP_ES,CLD,RET);
[U7Ø]   49:   }
[U7Ø]   5Ø: }
[ØØØ]   51:
[9X5]   52: /* initkeytable(): initialisiert die
[R93]   53:    Schlüsseltabelle*/
[ØØØ]   54:
[S73]   55: void initkeytable()
[GD1]   56: { word i;
[KU3]   57:   for(i=Ø;i<MAXENTRY;i++)
[YF3]   58:   { TOKEN(i)=C_ADR(i)=Ø;
[HK4]   59:     LEXEM(i)=ØL; EMIT(i)=dummy;
[U7Ø]   6Ø:   }
[N32]   61:   lastentry=Ø;
[U7Ø]   62: }
[ØØØ]   63:
[PT6]   64: /* lookup(string) : durchsucht Schlüssel-
[LH5]   65:    tabelle, ob string vorhanden ist;
[BF6]   66:    liefert bei Erfolg Schlüssel zurück,
[SN1]   67:    sonst -1; */
[ØØØ]   68:
[Y43]   69: int lookup(string)
[642]   7Ø: char *string;
[F91]   71: { int i;
[DØ4]   72:   for(i=Ø;i<lastentry;i++)
[VG4]   73:   if(!strcmp(string,LEXEM(i)))
[KM1]   74:   return(i);
[4B4]   75:   return(ENTRY_NOT_FOUND);
[U7Ø]   76: }
[ØØØ]   77:
[6X6]   78: #define getstring(nr) keytable[nr].lexem
[ØØØ]   79:
[LT6]   8Ø: /* installkey(string, tok) : trägt string,
[VN6]   81:    tok am Ende der Schlüsseltabelle ein */
[ØØØ]   82:
[2E5]   83: int installkey(string,tok,func)
[ØL1]   84: word tok;
[642]   85: char *string;
[792]   86: void (*func)();
[HA4]   87: { if(lastentry<MAXENTRY)
[M86]   88:   { if((LEXEM(lastentry)=malloc(strlen(
[Z62]   89:       string)+1))==Ø)
[LM4]   9Ø:     return(error(NO_MEMORY,""));
[DT3]   91:     TOKEN(lastentry)=tok;
[3M3]   92:     EMIT(lastentry)=func;
[7K5]   93:     strcpy(LEXEM(lastentry),string);
[HC3]   94:     return(lastentry++);
[U7Ø]   95:   }
[865]   96:   return(error(KEYTABLE_OVL,""));
[U7Ø]   97: }
[ØØØ]   98:
[6T5]   99: /* enterstring(string, tok) : trägt
[VR5]  1ØØ:    string, tok falls nicht vorhanden
[W76]  1Ø1:    in Schlüsseltabelle ein und liefert
[E65]  1Ø2:    immer den Schlüssel zurück */
[ØØØ]  1Ø3:
[ZX4]  1Ø4: int enterstring(string,tok)
[ØL1]  1Ø5: word tok;
[642]  1Ø6: char *string;
[EB4]  1Ø7: { int key=lookup(string);
[UB4]  1Ø8:   if(key==ENTRY_NOT_FOUND)
[KF6]  1Ø9:   return(installkey(string,tok,dummy));
[ZRØ]  11Ø:   else
[5Ø2]  111:   return(key);
[U7Ø]  112: }
[ØØØ]  113:
[UM2]  114: #define BATCOM 17
```

```
[ØØØ]   115:
[EL1]   116: /* Tokens */
[ZY4]   117: #define ALPHA   1  /* @ gefunden */
[LM6]   118: #define NEWLINE 2  /* Neue Zeile beginnt */
[TS4]   119: #define OP      3  /* Operator == */
[4T5]   12Ø: #define STRING  4  /* Zeichenketter*/
[AD5]   121: #define IDENT   5  /* Programmname */
[EW5]   122: #define BATVAR  6  /* Batch-Variable */
[ØØØ]   123:
[C53]   124: /* Batch-Kommandos */
[U83]   125: #define ECHO        Øx1ØØ
[Z83]   126: #define CALL        Øx1Ø1
[X93]   127: #define FOR         Øx1Ø2
[SB3]   128: #define GOTO        Øx1Ø3
[433]   129: #define IF          Øx1Ø4
[PD3]   13Ø: #define PAUSE       Øx1Ø5
[B93]   131: #define REM         Øx1Ø6
[PD3]   132: #define SHIFT       Øx1Ø7
[G43]   133: #define ON          Øx1Ø8
[K83]   134: #define OFF         Øx1Ø9
[UB3]   135: #define LABEL       Øx1ØA
[NB3]   136: #define NOT         Øx1Øb
[NØ4]   137: #define ERRORLEVEL  Øx1Øc
[HF3]   138: #define EXIST       Øx1Ød
[ØØØ]   139:
[VV3]   14Ø: char *strings[BATCOM]=
[8R5]   141: { "\n","==","@","ECHO","CALL","FOR","GOTO",
[9F5]   142:   "IF","PAUSE","REM","SHIFT","ON","OFF",
[V15]   143:   ":","NOT","ERRORLEVEL","EXIST"};
[EV7]   144: unsigned tokens[BATCOM] ={NEWLINE,OP,ALPHA,
[Y85]   145:          ECHO,CALL,FOR,GOTO,IF,PAUSE,REM,
[6S5]   146:          SHIFT,ON,OFF,LABEL,NOT,ERRORLEVEL,
[MG1]   147:          EXIST};
[CY6]   148: void (*e_funcs[BATCOM])()={dummy,dummy,
[6D5]   149:       dummy,dummy,dummy,dummy,dummy,
[6D5]   15Ø:       dummy,dummy,dummy,dummy,dummy,
[VB2]   151:       dummy,dummy};
[ØØØ]   152:
[XX1]   153: enterkeys()
[GD1]   154: { word i;
[G93]   155:   for(i=Ø;i<BATCOM;i++)
[FY5]   156:   installkey(strings[i],tokens[i],
[852]   157:                e_funcs[i]);
[U7Ø]   158: }
```

Listing 3. »dict.c« generiert das Wörterbuch für den Batch-Compiler Batch_It

```
[RS3]     1: /*Programm: batch_it.c
[4U4]     2:   Funktion: Semantik-Check und
[4D2]     3:             Codeerzeugung
[1R2]     4:   Sprache: Turbo C
[3L4]     5:   Autor: Matthias Oberdorfer
[T15]     6:   (c)1991 DMV Widuch GmbH & Co. KG */
[ØØØ]     7:
[8W2]     8: #include<stdio.h>
[7M2]     9: #include<dos.h>
[Ø43]    1Ø: #include<stdlib.h>
[YZ2]    11: #include"arg_do.c"
[VX2]    12: #include"asmdef.h"
[ØØØ]    13:
[ØS3]    14: #define FUN_EXT  ".fun"
[LM3]    15: #define COD_EXT  ".cod"
[XP3]    16: #define PRG_EXT  ".com"
[LP3]    17: #define PSP_SIZE Øx1ØØ
[EM3]    18: #define FUNC_SEG Øx1Ø3
[S66]    19: #define VERSION "Batch_It Ver.1.Ø ### \
[CN4]    2Ø:             1991 by Matthias Oberdorfer"
[ØØØ]    21:
[MM5]    22: /* Datei-Handle für Obj-Dateien */
[ØØØ]    23:
[957]    24: FILE *f_handle,*c_handle,*p_handle,*handle;
[SM6]    25: char f_name[13],c_name[13],p_name[13];
[AZ6]    26: unsigned f_count=Ø,c_count=Ø,p_count=Ø;
[ØØØ]    27:
[GX5]    28: /* emit_f(code) : gibt Zeichen in Datei
[W66]    29:    aus, die die Dateinummer handle hat.*/
[ØØØ]    3Ø:
[CX2]    31: void emit_f(code)
[XD2]    32: unsigned code;
[RC5]    33: { if(handle==f_handle) f_count++;
[4Z5]    34:   else if(handle==c_handle) c_count++;
[UP2]    35:   else p_count++;
[543]    36:   fputc(code,handle);
[U7Ø]    37: }
[ØØØ]    38:
[G35]    39: /* emit_str(string, len) : gibt
[5Ø5]    4Ø:    Zeichenkette der Länge len in
[2X1]    41:    handle aus */
[ØØØ]    42:
[TN3]    43: emit_str(string,len)
[HN3]    44: unsigned char *string;
[N72]    45: unsigned len;
[DA2]    46: { while(len--)
[ØC3]    47:   { emit_f(* string ++);
[U7Ø]    48:   }
[U7Ø]    49: }
[ØØØ]    5Ø:
```

```
[ØP3]    51: #define asm_it  arg_do
[B53]    52: #include<string.h>
[6L2]    53: #include"scan.c"
[ØØØ]    54:
[SC5]    55: /* open_com : öffnet Obj-Dateien
[5Ø4]    56:    Ø  : erfolgreich geöffnet
[G64]    57:    <Ø : Fehler beim öffnen */
[ØØØ]    58:
[W33]    59: int open_com(name)
[ØU1]    6Ø: char *name;
[7L6]    61: { unsigned point_pos=strcspn(name,".");
[872]    62:   if(point_pos)
[CT5]    63:   { strncpy(f_name,name,point_pos);
[NR3]    64:     f_name[point_pos]=Ø;
[U7Ø]    65:   }
[1C4]    66:   else strcpy(f_name,name);
[MS3]    67:   strcpy(c_name,f_name);
[1T3]    68:   strcpy(p_name,f_name);
[T14]    69:   strcat(f_name,FUN_EXT);
[GØ4]    7Ø:   strcat(c_name,COD_EXT);
[U24]    71:   strcat(p_name,PRG_EXT);
[L86]    72:   if((f_handle=fopen(f_name,"wb+"))==NULL)
[9E5]    73:     return(error(OPEN_FILE,f_name));
[A86]    74:   if((c_handle=fopen(c_name,"wb+"))==NULL)
[6E5]    75:     return(error(OPEN_FILE,c_name));
[H96]    76:   if((p_handle=fopen(p_name,"wb+"))==NULL)
[LE5]    77:     return(error(OPEN_FILE,p_name));
[TL1]    78:   return(Ø);
[U7Ø]    79: }
[ØØØ]    8Ø:
[BS2]    81: void close_com()
[Ø65]    82: { if(f_handle) fclose(f_handle);
[TZ4]    83:   if(c_handle) fclose(c_handle);
[Ø15]    84:   if(p_handle) fclose(p_handle);
[H65]    85:   unlink(f_name); unlink(c_name);
[U7Ø]    86: }
[ØØØ]    87:
[T63]    88: #define MAXGOTO 25Ø
[ØØØ]    89:
[161]    9Ø: struct
[GP2]    91: { word c_count;
[BH1]    92:   word key;
[R93]    93: }gotonr[MAXGOTO];
[PZ2]    94: word gotocount=Ø;
[DP2]    95: word incgoto=Ø;
[VZ1]    96: word adres;
[EE3]    97: unsigned save_str();
[ØØØ]    98:
[1YØ]    99: link()
[UC1]   1ØØ: { int ch;
[BN5]   1Ø1:   handle=f_handle; handle=p_handle;
[D36]   1Ø2:   asm_it(JMP,f_count&Øxff,f_count>>8);
[DV2]   1Ø3:   rewind(f_handle);
[DV4]   1Ø4:   while((ch=getc(f_handle))!=EOF)
[M52]   1Ø5:   { asm_it(ch);
[U7Ø]   1Ø6:   }
[3R5]   1Ø7:   asm_it(MOV_AH,Øx4a,MOV_BX,ØxØØ,Øx1Ø,
[A75]   1Ø8:          INT,Øx21,MOV_DX,FUNC_SEG&Øxff,
[S54]   1Ø9:          FUNC_SEG>>8,MOV_AH,Øx1a);
[AV2]   11Ø:   rewind(c_handle);
[GT1]   111:   c_count=Ø;
[235]   112:   while((ch=fgetc(c_handle))!=EOF)
[H44]   113:   { if(incgoto<gotocount)
[PT3]   114:     /* gibts noch includes */
[9W6]   115:     { if(c_count==gotonr[incgoto].c_count)
[R13]   116:       /* stelle von goto */
[X64]   117:       { if(!ch && !fgetc(c_handle))
[UC1]   118:         { adres=
[S35]   119:            C_ADR(gotonr[incgoto++].key)+
[7H1]   12Ø:            p_count+
[ØP1]   121:            PSP_SIZE-
[5M4]   122:            gotonr[incgoto].c_count;
[WH4]   123:            asm_it(adres&Øxff,adres>>8);
[MS1]   124:            c_count++;
[U7Ø]   125:         }
[ZRØ]   126:         else
[WR5]   127:         {printf("Error ch %x incgoto %x\n",
[VW1]   128:                 ch,incgoto);
[961]   129:          exit(Ø);
[U7Ø]   13Ø:         }
[U7Ø]   131:       }
[R93]   132:       else { asm_it(ch); }
[U7Ø]   133:     }
[7G3]   134:     else{ asm_it(ch); }
[5Y1]   135:     c_count ++;
[U7Ø]   136:   }
[LZ1]   137:   asm_it(RET);
[U7Ø]   138: }
[ØØØ]   139:
[4K3]   14Ø: gib_aus(adr,adr_len)
[3W2]   141: word adr,adr_len;
[Y43]   142: { handle=c_handle;
[GN5]   143:   asm_it(MOV_AH,Øx4Ø,MOV_BX,Øx Ø1,ØxØØ,
[DM3]   144:          MOV_CX,Ø,Ø,MOV_RLM,CL,
[EØ4]   145:          (adr_len+FUNC_SEG)&Øxff,
[VV4]   146:          (adr_len+FUNC_SEG)>>8,INC_CX,
[6G2]   147:          INC_CX,MOV_DX,
[A83]   148:          (adr+FUNC_SEG)&Øxff,
```

Listing 4. »batch_it.c« führt den Syntaxcheck und die Codegenerierung durch

```
[K94]  149:          (adr+FUNC_SEG)>>8,INT,Øx21);
[U7Ø]  15Ø: }
[ØØØ]  151:
[4T2]  152: /* Zählt args %p */
[WP3]  153: word count_para(str)
[7L1]  154: char *str;
[B82]  155: { word i,para;
[GD5]  156:   for(i=Ø,para=Ø;i<strlen(str);i++)
[EM5]  157:   if(*(str+i)=='%' && isdigit(*(str+i+1)))
[F51]  158:     para++;
[H52]  159:   return(para);
[U7Ø]  16Ø: }
[1N1]  161: word adr2;
[ØØØ]  162:
[L93]  163: word para=Ø,s_adr=Ø;
[7P1]  164: para_arg()
[ZR1]  165: { if(!para)
[7S5]  166:   { handle=f_handle; s_adr=f_count;
[R64]  167:     asm_it(Ø,Ø); para=f_count;
[K63]  168:     /* CX: String laenge *
[Z81]  169:        SI: ADR
[9U3]  17Ø:        Rück CX neue Laenge */
[ØD5]  171:     asm_it(PUSH_CX,CMP_CX,Ø,JZ,Øx2e,
[PØ5]  172:            MOV_RLM,AL_SI,CMP_AL,ØxØa,JA,
[K95]  173:            Øx24,MOV_BX,Ø,Ø,MOV_RLRL,BL_AL,
[312]  174:            CLC,ADC_L,BL,
[1P3]  175:            (s_adr+FUNC_SEG)&Øxff,
[HB3]  176:            (s_adr+FUNC_SEG)>>8,
[6Z4]  177:            CALL_F,Øx1d,ØxØØ,MOV_RLML,Øx2Ø,
[4F5]  178:            MOV_RLRL,AL_SI,CMP_DX,Ø,JZ,ØxØe,
[PG4]  179:            CALL_F,Øx56,ØxØØ,CALL_F,Øx7Ø,
[P15]  18Ø:            ØxØØ,POP_BX,CLC,ADC,BX_DX,ADC,
[B45]  181:            CX_DX,DEC_BX,PUSH_BX,INC_SI,
[5L4]  182:            DEC_CX,JNZ,Øxd2,POP_CX,RET);
[ØØØ]  183:
[7U3]  184:     /* U1 : sucht nach arg nr :
[BK3]  185:        dann DI: adr DX : len*/
[3B6]  186:     asm_it(MOV_DX,Ø,Ø,MOV_RLM,DL,Øx8Ø,ØxØØ,
[UM4]  187:            MOV_DI,Øx81,ØxØØ,CMP_DX,Ø,JZ,
[RD4]  188:            Øx1a,MOV_RLM,AL_DI,DEC_DX,
[814]  189:            INC_DI,CMP_AL,' ',JZ,Øxf3,
[TY4]  19Ø:            DEC_BX,JZ,Øx1Ø,CMP_DX,Ø,JZ,Øx0a,
[3Y4]  191:            DEC_DX,MOV_RLM,AL_DI,INC_DI,
[3T4]  192:            CMP_AL,' ',JNZ,Øxf3,JMP_R,Øxe1,
[DE4]  193:            RET,DEC_DI,PUSH_DI,INC_DI,
[K64]  194:            PUSH_CX,MOV_CX,Øx01,ØxØØ,
[EB5]  195:            CMP_DX,Ø,JZ,Øx0a,MOV_RLM,AL_DI,
[R75]  196:            DEC_DX,INC_DI,INC_CX,CMP_AL,' ',
[X45]  197:            JNZ,Øxf2,DEC_CX,MOV_MR,DX_CX,
[UD3]  198:            POP_CX,POP_DI,RET);
[ØØØ]  199:
[F56]  2ØØ:     /* U2 : kopiert reststr nach rechts */
[3L5]  2Ø1:     asm_it(CMP_DX,Ø,JZ,Øxfa,PUSH_CX,
[KW5]  2Ø2:            PUSH_DI,PUSH_SI,PUSH_DX,DEC_DX,
[4E5]  2Ø3:            INC_CX,INC_CX,MOV_MR,DI_SI,CLC,
[R45]  2Ø4:            ADC,DI_CX,ADC,DI_DX,ADC,SI_CX,
[375]  2Ø5:            STD,REPZ,MOVSB,POP_DX,POP_SI,
[CD3]  2Ø6:            POP_DI,POP_CX,RET);
[ØØØ]  2Ø7:
[G83]  2Ø8:     /* U3 : kopiert arg */
[3L5]  2Ø9:     asm_it(CMP_DX,Ø,JZ,Øxfa,PUSH_CX,
[GM4]  21Ø:            PUSH_DX,PUSH_DI,PUSH_SI,
[NU4]  211:            MOV_MR,CX_DX,MOV_MR,DX_SI,
[YB5]  212:            MOV_MR,SI_DI,MOV_MR,DI_DX,CLD,
[CU5]  213:            REPZ,MOVSB,POP_SI,POP_DI,POP_DX,
[GC3]  214:            POP_CX,DEC_CX,RET);
[U7Ø]  215:   }
[U7Ø]  216: }
[ØØØ]  217:
[FZ4]  218: interpret_para(adr2,adr_len)
[Z13]  219: word adr2,adr_len;
[E93]  22Ø: { unsigned adr=adr2;
[GS2]  221:   handle=c_handle;
[AN5]  222:   asm_it(MOV_SI,(adr+FUNC_SEG)&Øxff,
[6X2]  223:          (adr+FUNC_SEG)>>8,
[DM3]  224:          MOV_CX,Ø,Ø,MOV_RLM,CL,
[EØ4]  225:          (adr_len+FUNC_SEG)&Øxff,
[EX4]  226:          (adr_len+FUNC_SEG)>>8,MOV_BX,
[1C3]  227:          (para+FUNC_SEG)&Øxff,
[F23]  228:          (para+FUNC_SEG)>>8,
[CN3]  229:          CALL_BX,MOV_RLRL,CL,
[EØ4]  23Ø:          (adr_len+FUNC_SEG)&Øxff,
[DW3]  231:          (adr_len+FUNC_SEG)>>8);
[U7Ø]  232: }
[ØØØ]  233:
[C51]  234: shift()
[Y43]  235: { handle=c_handle;
[GR6]  236:   asm_it(MOV_RLM,CL,(s_adr+FUNC_SEG)&Øxff,
[HB3]  237:          (s_adr+FUNC_SEG)>>8,
[ØE3]  238:          INC_CL,MOV_RLRL,CL,
[1P3]  239:          (s_adr+FUNC_SEG)&Øxff,
[4F3]  24Ø:          (s_adr+FUNC_SEG)>>8);
[U7Ø]  241: }
[ØØØ]  242:
[DB6]  243: /* save_str(str, t) : Speichert Str in
[MR4]  244:    Funtionsegment ab im Format
[R82]  245:    1.Byte: Länge
[AD2]  246:    n.Byte: String
[LS4]  247:    n+1,n+2. Byte wenn t=Ø:'\r', '\n'
[KP2]  248:    sonst    '\Ø', '\n'
[FN4]  249:    liefert Adrim Funcsegment */
```

```
[ØØØ]  25Ø:
[E42]  251: unsigned i,t;
[YC4]  252: unsigned save_str(str,te)
[7L1]  253: char *str;
[KD1]  254: word te;
[6C3]  255: { word adr=f_count;
[BF2]  256:   t=te; adr2=adr;
[LS2]  257:   handle=f_handle;
[H16]  258:   asm_it(strlen(str)-count_para(str));
[SW2]  259:   /* Ausgabe string */
[B84]  26Ø:   for(i=Ø;i<strlen(str);i++)
[EM5]  261:   if(*(str+i)=='%' && isdigit(*(str+i+1)))
[CXØ]  262:   { i++;
[C33]  263:     asm_it(*(str+i)-'Ø');
[U7Ø]  264:   }
[ZRØ]  265:   else
[FN4]  266:   if(*(str+i)=='%' && *(str+i+1)=='%')
[CXØ]  267:   { i++;
[5P2]  268:     asm_it(*(str+i));
[U7Ø]  269:   }
[ZRØ]  27Ø:   else
[633]  271:   { asm_it(*(str+i));}
[YB4]  272:   if(!t) { asm_it('\r','\n');}
[V64]  273:   else { asm_it('\Ø','\n');}
[B54]  274:   /* Platz für arguments */
[H63]  275:   if(count_para(str))
[NL2]  276:     for(i=Ø;i<64; i++)
[FT2]  277:     { asm_it('M','O');
[U7Ø]  278:     }
[AS5]  279:   para_arg(); /* einbinden PARA_FUNC */
[4Z1]  28Ø:   return(adr);
[U7Ø]  281: }
[ØØØ]  282:
[UF5]  283: word echo_on   = 1; /* ECHO ON ¦OFF*/
[UD5]  284: word echo_line = 1;  /* Zeile @ ¦! */
[9Y6]  285: word this_line=Ø;  /* Zeile gespeichert?*/
[ØØØ]  286:
[DWØ]  287: echo()
[EP3]  288: { word lp=linepos,adr;
[HN4]  289:   word key=TOKEN(gettoken(Ø));
[HL2]  29Ø:   if(key!=NEWLINE)
[B15]  291:   if(TOKEN(gettoken(Ø))==NEWLINE)
[WX3]  292:   { if(key==ON) echo_on=1;
[4P4]  293:     else if(key==OFF) echo_on=Ø;
[ZRØ]  294:     else
[WG5]  295:     { adr=save_str(&linebuf[lp+1],Ø);
[VC4]  296:       interpret_para(adr+1,adr);
[Z53]  297:       gib_aus(adr+1,adr);
[U7Ø]  298:     }
[UL1]  299:     return(1);
[U7Ø]  3ØØ:   }
[2D5]  3Ø1:   adr=save_str(&linebuf[lp+1],Ø);
[VC4]  3Ø2:   interpret_para(adr+1,adr);
[Z53]  3Ø3:   gib_aus(adr+1,adr);
[U7Ø]  3Ø4: }
[MR5]  3Ø5: char *Pausestr="Wenn bereit, eine "
[984]  3Ø6:           "beliebige Taste drücken";
[C92]  3Ø7: word p_adr=Ø;
[G51]  3Ø8: pause()
[2W6]  3Ø9: { if(!p_adr) p_adr=save_str(Pausestr,Ø);
[145]  31Ø:   interpret_para(p_adr+1,p_adr);
[FZ3]  311:   gib_aus(p_adr+1,p_adr);
[GS2]  312:   handle=c_handle;
[MH4]  313:   asm_it(MOV_AH,Øx08,INT,Øx21);
[U7Ø]  314: }
[X61]  315: _goto()
[G34]  316: { if(gotocount<MAXGOTO)
[Y43]  317:   { handle=c_handle;
[YR2]  318:     asm_it(MOV_BX);
[XC6]  319:     gotonr[gotocount].c_count=c_count;
[K46]  32Ø:     gotonr[gotocount].key=gettoken(Ø);
[Z12]  321:     gotocount++;
[D53]  322:     asm_it(Ø,Ø,JMP_BX);
[U7Ø]  323:   }
[U7Ø]  324: }
[CW3]  325: word adr_if,str1,str2;
[7R1]  326: word nr=Ø;
[MZ2]  327: char *label ="%%ØØØØ";
[GS2]  328: char string[13Ø];
[VYØ]  329: _if_()
[3S2]  33Ø: { word key,not=Ø;
[YØ5]  331:   if(TOKEN(key=gettoken(1))==NOT)
[V91]  332:   { not=1;
[AN2]  333:     key=gettoken(1);
[U7Ø]  334:   }
[1Y2]  335:   switch(TOKEN(key))
[F23]  336:   { case ERRORLEVEL:
[7N2]  337:       key=gettoken(Ø);
[925]  338:       adr_if=save_str(LEXEM(key),1);
[SH5]  339:       interpret_para(adr_if+1,adr_if);
[GS2]  34Ø:       handle=c_handle;
[V95]  341:       asm_it(MOV_AX,ØxØØ,Øx4d,INT,Øx21,
[NS4]  342:              MOV_BX,Øxff,Øxff,CLC,ADC_AL,
[T12]  343:              'Ø',CMP_AL_M,
[WØ4]  344:              (adr_if+1+FUNC_SEG)&Øxff,
[AR3]  345:              (adr_if+1+FUNC_SEG)>>8,
[BZ3]  346:              JZ,Øx03,MOV_BX,ØxØØ,ØxØØ);
[PØ1]  347:     break;
[WØ2]  348:     case EXIST:
[7N2]  349:       key=gettoken(Ø);
[925]  35Ø:       adr_if=save_str(LEXEM(key),1);
```

```
[SH5]  351:        interpret_para(adr_if+1,adr_if);
[GS2]  352:        handle=c_handle;
[585]  353:        asm_it(MOV_AX,Øx ØØ,Øx3d,MOV_DX,
[WØ4]  354:               (adr_if+1+FUNC_SEG)&Øxff,
[AR3]  355:               (adr_if+1+FUNC_SEG)>>8,
[EW2]  356:               MOV_BX,Øxff,Øxff,
[FP4]  357:               INT,Øx21,JAE,Øxøa,MOV_MR,AX,
[C34]  358:               MOV_AX,Øx ØØ,Øx3e,INT,Øx21,
[HV1]  359:               MOV_BX,Ø,Ø);
[PØ1]  36Ø:      break;
[8E1]  361:      default:
[Z36]  362:        while(linebuf2[tokpos]==' ')tokpos++;
[S16]  363:        strncpy(string,&linebuf2[tokpos],
[XV2]  364:                linepos-tokpos);
[ØK4]  365:        string[linepos-tokpos]=Ø;
[L54]  366:        str1=save_str(string,1);
[VR4]  367:        interpret_para(str1+1,str1);
[7N2]  368:        key=gettoken(Ø);
[7N2]  369:        key=gettoken(Ø);
[Z36]  37Ø:        while(linebuf2[tokpos]==' ')tokpos++;
[S16]  371:        strncpy(string,&linebuf2[tokpos],
[XV2]  372:                linepos-tokpos);
[ØK4]  373:        string[linepos-tokpos]=Ø;
[M54]  374:        str2=save_str(string,1);
[2S4]  375:        interpret_para(str2+1,str2);
[6D5]  376:        asm_it(MOV_BX,Øxff,Øxff,MOV_SI,
[PD3]  377:               (str1+FUNC_SEG)&Øxff,
[443]  378:               (str1+FUNC_SEG)>>8,
[AT4]  379:               MOV_DI,(str2+FUNC_SEG)&Øxff,
[643]  38Ø:               (str2+FUNC_SEG)>>8,
[AZ4]  381:               INC_DI,INC_SI,MOV_RLM,CL_SI,
[FY4]  382:               CMP_CL,Ø,JZ,Øx Ø7,CMP_CL_DI,JZ,
[RP2]  383:               Øxf3,MOV_BX,Ø,Ø);
[PØ1]  384:      break;
[U7Ø]  385:    }
[DT4]  386:    if(not) { asm_it(NOT_BX); }
[AN5]  387:    asm_it(CMP_BX,Ø,JNZ,Øx Ø5,MOV_BX);
[XC6]  388:    gotonr[gotocount].c_count=c_count;
[DR3]  389:    nr=bat[act_f].zeile;
[6B3]  39Ø:    *(label+2)=(nr/1ØØØ)+'Ø';
[2K2]  391:    nr -= (nr/1ØØØ)*1ØØØ;
[V63]  392:    *(label+3)=(nr/1ØØ)+'Ø';
[WD2]  393:    nr -= (nr/1ØØ)*1ØØ;
[S63]  394:    *(label+4)=(nr/1Ø)+'Ø';
[632]  395:    nr -= (nr/1Ø)*1Ø;
[2G2]  396:    *(label+5)=nr+'Ø';
[9B7]  397:    gotonr[gotocount].key=enterstring(label,
[F61]  398:                                      IDENT);
[Z12]  399:    gotocount++;
[D53]  4ØØ:    asm_it(Ø,Ø,JMP_BX);
[U7Ø]  4Ø1: }
[ØØØ]  4Ø2:
[5Y1]  4Ø3: void parse()
[4N1]  4Ø4: { int key;
[M62]  4Ø5:   unsigned adr;
[LS2]  4Ø6:   handle=f_handle;
[S56]  4Ø7:   emit_str(VERSION,strlen(VERSION));
[FF6]  4Ø8:   for(key=strlen(VERSION); key<13Ø; key++)
[A92]  4Ø9:   { emit_f('M');
[U7Ø]  41Ø:   }
[3H5]  411:   while((key=gettoken(Ø))!=EOF_SCAN)
[TØ4]  412:   { if(key>=Ø && !C_ADR(key))
[VS3]  413:     { C_ADR(key)=f_count;
[GT1]  414:       EMIT(key)();
[U7Ø]  415:     }
[R45]  416:     if(TOKEN(key)!=ALPHA && echo_on &&
[3G5]  417:        echo_line && TOKEN(key)!=NEWLINE &&
[875]  418:        linesize>Ø && TOKEN(key)!=LABEL)
[3N4]  419:     { echo_line=Ø; this_line=1;
[944]  42Ø:       adr=save_str(linebuf,Ø);
[VC4]  421:       interpret_para(adr+1,adr);
[Z53]  422:       gib_aus(adr+1,adr);
[U7Ø]  423:     }
[1Y2]  424:     switch(TOKEN(key))
[P12]  425:     { case ALPHA:
[KØ2]  426:         echo_line=Ø;
[PØ1]  427:       break;
[9M1]  428:       case ECHO:
[T31]  429:         echo();
[U43]  43Ø:         linepos=linesize;
[PØ1]  431:       break;
[BP1]  432:       case GOTO:
[S81]  433:         _goto();
[U43]  434:         linepos=linesize;
[PØ1]  435:       break;
[7A1]  436:       case IF:
[761]  437:         _if_();
[PØ1]  438:       break;
[CZ1]  439:       case SHIFT:
[771]  44Ø:         shift();
[U43]  441:         linepos=linesize;
[PØ1]  442:       break;
[5F1]  443:       case REM:
[U43]  444:         linepos=linesize;
[PØ1]  445:       break;
[ØY1]  446:       case PAUSE:
[B71]  447:         pause();
[U43]  448:         linepos=linesize;
[PØ1]  449:       break;
[UW1]  45Ø:       case LABEL:
[7N2]  451:         key=gettoken(Ø);
```

```
[SB3]  452:         C_ADR(key)=c_count;
[U43]  453:         linepos=linesize;
[PØ1]  454:       break;
[L62]  455:       case NEWLINE:
[PØ2]  456:         echo_line=1;
[722]  457:         this_line=Ø;
[DR3]  458:         nr=bat[act_f].zeile;
[6B3]  459:         *(label+2)=(nr/1ØØØ)+'Ø';
[2K2]  46Ø:         nr -= (nr/1ØØØ)*1ØØØ;
[V63]  461:         *(label+3)=(nr/1ØØ)+'Ø';
[WD2]  462:         nr -= (nr/1ØØ)*1ØØ;
[S63]  463:         *(label+4)=(nr/1Ø)+'Ø';
[632]  464:         nr -= (nr/1Ø)*1Ø;
[2G2]  465:         *(label+5)=nr+'Ø';
[3X5]  466:         if(lookup(label)!=ENTRY_NOT_FOUND)
[1A3]  467:          C_ADR(lookup(label,
[1R2]  468:                       IDENT))=c_count;
[PØ1]  469:       break;
[8E1]  47Ø:       default:
[3F5]  471:       /* Ausgabe der Zeile in Func-teil */
[N92]  472:         if(!this_line)
[996]  473:         { adr=save_str(&linebuf[tokpos],Ø);
[VC4]  474:          interpret_para(adr+1,adr);
[U7Ø]  475:         }
[CV4]  476:         /* und Aufruf in den Code-teil */
[GS2]  477:         handle=c_handle;
[AN5]  478:         asm_it(MOV_SI,(adr+FUNC_SEG)&Øxff,
[A54]  479:                (adr+FUNC_SEG)>>8,MOV_BX,
[LD3]  48Ø:                (exec+FUNC_SEG)&Øxff,
[4H4]  481:                (exec+FUNC_SEG)>>8,CALL_BX);
[1Y5]  482:         linepos=linesize;/* zeile beendet *
       /
[PØ1]  483:       break;
[U7Ø]  484:     }
[U7Ø]  485:   }
[U7Ø]  486: }
[ØØØ]  487:
[5G2]  488: main(argc,argv)
[2E2]  489: unsigned argc;
[R42]  49Ø: char *argv[];
[T24]  491: { printf("%s\n",VERSION);
[RY2]  492:   ARG_FUNC=emit_f;
[5F1]  493:   if(argc>1)
[1V2]  494:   { initkeytable();
[SZ1]  495:     enterkeys();
[SG3]  496:     if(open_bat(*(argv+1)))
[AØ4]  497:     { if(!open_com(*(argv+1)))
[KU2]  498:       { parse(); link();
[434]  499:         close_com(); close_bat();
[XR4]  5ØØ:         printf("Counts func_seg: %x "
[HF2]  5Ø1:                "code_seg : %x  "
[BF2]  5Ø2:                "prg_seg: %x\n",
[XN4]  5Ø3:                f_count,c_count,p_count);
[U7Ø]  5Ø4:       }
[U7Ø]  5Ø5:     }
[W35]  5Ø6:     else error(OPEN_FILE,*(argv+1));
[U7Ø]  5Ø7:   }
[ZRØ]  5Ø8:   else
[LZ3]  5Ø9:   { bat[++ act_f].zeile=Ø;
[NV2]  51Ø:     error(NO_FILE,"");
[U7Ø]  511:   }
[U7Ø]  512: }
```

Listing 4. »batch__it.c« führt den Syntaxcheck und die Codegenerierung durch

```
[UM2]    1: /* Datei: asmdef.h
[936]    2:    Funktion: definiert Assembkercodes
[1R2]    3:    Sprache: Turbo C
[3L4]    4:    Autor: Matthias Oberdorfer
[T15]    5:    (c)1991 DMV Widuch GmbH & CO. KG */
[ØØØ]    6:
[GR2]    7: #define CS      Øx2e
[FD3]    8: #define PUSH_CS Øxøe
[V63]    9: #define POP_DS  Øx1f
[A63]   1Ø: #define POP_ES  Øxø7
[3E3]   11: #define PUSH_CX Øx51
[7E3]   12: #define PUSH_DX Øx52
[GC3]   13: #define PUSH_DI Øx57
[WD3]   14: #define PUSH_SI Øx56
[A73]   15: #define POP_SI  Øx5e
[2E3]   16: #define PUSH_BX Øx53
[K73]   17: #define POP_BX  Øx5b
[D63]   18: #define POP_DI  Øx5f
[B73]   19: #define POP_CX  Øx59
[N73]   2Ø: #define POP_DX  Øx5a
[ØØ3]   21: #define INT     Øxcd
[CZ2]   22: #define JMP     Øxe9
[S43]   23: #define JMP_R   Øxeb
[PS2]   24: #define JZ      Øx74
[VY2]   25: #define JNZ     Øx75
[CP2]   26: #define JB      Øx72
[EP2]   27: #define JA      Øx77
[YU2]   28: #define JAE     Øx73
[6Z2]   29: #define RET     Øxc3
[1Y2]   3Ø: #define NOP     Øx9Ø
[S83]   31: #define MOV_MS  Øx8c
[MH3]   32: #define MOV_RLML Øxbø
[B83]   33: #define MOV_MR  Øx89
```

Listing 5. »asmdef.h«: Definition der Assembler- Codes

```
[EK3]    34: #define MOV_RLRL Øx88
[Ø43]    35: #define BL_AL    Øxc3
[G63]    36: #define DX_CX    Øxca
[Z53]    37: #define DX_SI    Øxf2
[M53]    38: #define SI_DI    Øxfe
[363]    39: #define CX_DX    Øxd1
[K23]    4Ø: #define CL_SI    ØxØc
[R43]    41: #define DI_SI    Øxf7
[853]    42: #define DI_CX    Øxcf
[Y43]    43: #define DI_DX    Øxd7
[563]    44: #define SI_CX    Øxce
[M83]    45: #define MOV_SM   Øx8e
[F83]    46: #define MOV_RM   Øx8b
[ZD3]    47: #define MOV_RLM Øx8a
[Z13]    48: #define AL_SI    ØxØ4
[H13]    49: #define AL_DI    ØxØ5
[5G3]    5Ø: #define BX_SI_P Øx5c
[363]    51: #define BX_DX    Øxd3
[553]    52: #define CMP_AL   Øx3c
[ND4]    53: #define CMP_CL_DI Øx3a,ØxØd
[G94]    54: #define CMP_AL_M  Øx3a,ØxØ6
[AX3]    55: #define CMP_CL   Øx8Ø,Øxf9
[3Z3]    56: #define CMP_DX   Øx83,Øxfa
[1Z3]    57: #define CMP_BX   Øx83,Øxfb
[HY3]    58: #define CMP_CX   Øx83,Øxf9
[914]    59: #define NOT_BX   Øxf7,Øxd3
[AX2]    6Ø: #define CLC      Øxf8
[KZ2]    61: #define STD      Øxfd
[TX2]    62: #define CLD      Øxfc
[LT2]    63: #define ADC      Øx11
[BY2]    64: #define ADC_L    Øx12
[V13]    65: #define ADC_AL   Øx14
[WØ3]    66: #define REPZ     Øxf3
[K33]    67: #define MOVSB    Øxa4
[XR2]    68: #define SS       Øx16
[SR2]    69: #define SP       Øx26
[RR2]    7Ø: #define CX       ØxØe
[DT2]    71: #define AX       Øxc3
[AP2]    72: #define DL       Øx16
[LP2]    73: #define CL       ØxØe
[NP2]    74: #define BL       Øx1e
[4Ø3]    75: #define STI      Øxfb
[BY2]    76: #define CLI      Øxfa
[563]    77: #define CALL_F   Øxe8
```

```
[X44]    78: #define CALL_BX Øxff,Øxd3
[VØ4]    79: #define JMP_BX  Øxff,Øxe3
[G93]    8Ø: #define MOV_SI  Øxbe
[3A3]    81: #define MOV_SP  Øxbc
[S73]    82: #define MOV_AH  Øxb4
[X73]    83: #define MOV_CH  Øxb5
[C93]    84: #define MOV_AX  Øxb8
[H93]    85: #define MOV_CX  Øxb9
[V93]    86: #define MOV_DX  Øxba
[S93]    87: #define MOV_BX  Øxbb
[G93]    88: #define MOV_SI  Øxbe
[L83]    89: #define MOV_DI  Øxbf
[U43]    9Ø: #define INC_CX  Øx41
[X33]    91: #define DEC_CX  Øx49
[443]    92: #define DEC_BX  Øx4b
[743]    93: #define DEC_DX  Øx4a
[Z23]    94: #define DEC_DI  Øx4f
[K43]    95: #define INC_SI  Øx46
[N33]    96: #define INC_DI  Øx47
[ZX3]    97: #define INC_CL  Øxfe,Øxc1
```

Listing 5. »asmdef.h«: Definition der Assembler-Codes

```
[ES2]     1: /*Datei: arg_do.c
[G96]     2:   Funktion:Verarbeitung der variablen
[1W2]     3:             Argumentenliste
[1R2]     4:   Sprache: Turbo C
[3L4]     5:   Autor: Matthias Oberdorfer
[T15]     6:   (c)1991 DMV Widuch GmbH & Co. KG */
[ØØØ]     7:
[RR4]     8: void (*ARG_FUNC)(unsigned c);
[ØØØ]     9:
[655]    1Ø: #define arg_do _BP=_SP;arg_exe
[ØØØ]    11:
[1H3]    12: void far arg_exe(arg)
[182]    13: unsigned arg;
[BN5]    14: { unsigned r_bp,argc,*argv=&arg,i;
[CZ1]    15:   r_bp=_BP+6;
[MT5]    16:   argc=((*((unsigned *)&arg-3)-r_bp)>>1);
[X33]    17:   for(i=Ø; i<argc; i++)
[G43]    18:   ARG_FUNC(*(argv+i));
[U7Ø]    19: }
```

Listing 6. »arg__do.c« verarbeitet die Argumentenliste

Impressum

Herausgeber: Christian Widuch
Redaktionsdirektor: Volker Everts

Redaktion

Chefredakteur: Ulrich Eike (ue)

Chefin vom Dienst: Gabi Börger (gb)

Ressort Aktuelles: Gabi Börger (gb) (Ltg.), Gerhard Brauer (ge)
Ressort Anwendung: Marina Baader (ba), Alois Kneisle (al), Herwig Weihe (hw), Gerda Hutt (gh)

Ressort Hardware: Wolfgang Hofner (wh) (Ltg.), Ralph Kunze (rk), Karl-Theo Hartmann (kh)

Ressort Programmierung: Dieter Bäumler (db) (Ltg.), Sebastian Kranz (sk), Ralf Morgenstern (rm), Wolfgang Nefzger (wn), Erhard Thomas (et), Kerrin Uhlmann (uk)

Redaktionsmarketing: Ingo Everts (ie)

Schlußredaktion: Karin Adlmannseder (ka), Annette Baumhof (ab)

Redaktionsassistenz: Angelika Albrecht, Anke Ludwig, Helga Menne

Layout: Michaela Küppers, Cristiana Seiser

Fotografie: Heinz Harcuba

Leitung des Redaktionsbüros: Volker Everts

Anschrift Redaktion: Redaktion DOS International, Gruberstr. 46c, 8011 Poing bei München, Telefon: (08121) 769-0, Telefax: (08121) 79046, Btx: *dmv # oder *20072 #

Sie erreichen die Redaktion werktags telefonisch von 15 bis 17 Uhr.

Manuskripte, Programme: bitte an die Redaktion DOS International, Gruberstr. 46c, 8011 Poing bei München, senden.

Für unverlangt eingesandte Manuskripte und Datenträger sowie Fotos übernehmen der Verlag und die Redaktion keine Haftung. Die Zustimmung zum Abdruck wird vorausgesetzt. Das Urheberrecht für veröffentlichte Manuskripte liegt ausschließlich beim Verlag. Nachdruck sowie Vervielfältigung oder sonstige Verwertung von Texten nur mit schriftlicher Genehmigung des Verlages. Für veröffentlichte Programme und Basteleien übernehmen wir weder Gewähr noch Haftung. Schaltungen und verwendetet Bezeichnungen müssen nicht frei von gewerblichen Schutzrechten sein. Namentlich gekennzeichnete Fremdbeiträge geben nicht in jedem Fall die Meinung der Redaktion wieder.

Anzeigenverkauf

Anzeigenleitung: Wolfgang Brill
Anzeigenverkauf für PLZ 1, 4, 5:
Gerlinde Rachow (0 56 51) 80 93 90
Sylvia Stephani (0 56 51) 80 93 80
Bernd Heckmann (0 56 51) 80 93 81
Karina Ehrlich (0 56 51) 80 93 71
Telefax (0 56 51) 80 94 44

Anzeigenverkauf für PLZ 2 + 3
DMV-Verlagsbüro Hamburg, Ohlsdorfer Straße 34, 2000 Hamburg 60, Telefon: (0 40) 46 12 33, Telefax: (0 40) 47 43 10
Leitung: Sylvia Ehrenpfordt
Anzeigenverkauf: Sabine Bindseil, Ralph Streng
Anzeigenverkauf für PLZ 6 – 8
DMV-Verlagsbüro München, Zaunkönigweg 2c, 8000 München 82, Telefon: (0 89) 4 39 10 87, Telefax: (0 89) 4 39 10 80
Leitung: Britta Fiebig
Anzeigenverkauf: Peter Schätzle, Hannelore David, Ilona Sehm

Repräsentanten im Ausland:
Great Britain
Huson European Media, Gerry Rhoades-Brown, 10-11 The green Business Centre, The Causeway, Staines, Middlesex, TW18 3AL, Telephone No: 0784 46 99 00 Fax No: 0784 46 99 96

Taiwan
PRISCO, Mr. Parson Lee, 2F, No.17, Alley 7, Lane 217, Chung Hsiao E. Rd. Sec. 3, Taipei, Taiwan, R.O.C, Phone: [886]-2-711 48 33-6, Fax: [886]-2-741 51 10,
Hong Kong
SET WAY Promotion Ltd., Mr. James Huang, Flat 13, 19/f, Block U, Telford Gardens, Kowloon Bay, Kowloon, Hong Kong, Phone [852]-799 83 64, Fax [852]-795 15 64

Singapore
Major Media Private Ltd., attn. Mr. Hoo Siew-Sai, 95 South Bridge Road, #09-13 Pidemco Centre, Singapore 0105, Phone [65]-532 40 26, Fax [65]-532 40 27

Anzeigenverwaltung
Andrea Giese, Beate Kranz

Druckunterlagen-Disposition
Christina Wabra (05651/809-372)

Verlag

Produktionsleitung: Uwe Siebert

Koordination: Martina Gress

Fotosatz: Claudia Ebbrecht (Ltg.), Dirk Anhof

Montage/Reprografie: Margarete Schenk (Ltg.), Helmut Skoupy (Ltg.), Manuela Eska, Andrea Gundlach

Werbeleitung: Jutta Himmelmann

Werbegrafik: Mohammed Hawa, Uwe Kalischinsky

Anschrift Verlag: DMV Daten- und Medien-Verlag Widuch GmbH & Co. KG, Fuldaer Str. 6, 3440 Eschwege, Telefon: (05651) 809-0, Telefax: (05651) 809-333

Vertrieb: Verlagsunion Erich Pabel-Arthur Moewig KG (VPM) Friedrich-Bergius-Straße 20, 6200 Wiesbaden

Druck: Druckerei Jungfer, 3420 Herzberg

Bezugspreise
DOS Extra erscheint dreimonatlich.
Einzelpreis DM 18,– /sfr. 18,– /öS 145,–

Abonnementpreise
Die Preise verstehen sich grundsätzlich einschließlich Porto und Verpackung.

Inland:
Bezugspreis 4 Ausgaben: DM 60,–
Europäisches Ausland:
Bezugspreis 4 Ausgaben: DM 70,–
Außereuropäisches Ausland:
Bezugspreis 4 Ausgaben: DM 80,–

Bankverbindungen:
Postscheck Frankfurt/M: Kto.-Nr.: 23043-608
Raiffeisenbank Eschwege:
BLZ: 522 603 85, Kto.-Nr.: 245 7008

Die Abonnementbestellung kann innerhalb einer Woche nach Auftrag beim DMV-Verlag, Postfach 250, 3440 Eschwege, schriftlich widerrufen werden. Zur Wahrung der Frist reicht der Poststempel. Das Abonnement verlängert sich automatisch um 6 bzw. 12 Monate, wenn es nicht mindestens 6 Wochen vor Ablauf beim Verlag schriftlich gekündigt wird.

Na, endlich!

Ausgabe Nr. 9/91 7,

Das verständliche Compute

PC pur & TEST MAGAZIN

Die besten

9 September 91
4. Jahrgang
ISSN Nr. 0935-9281
Eine Publikation von DMV

DRUCKER

Orientierungshilfe:
Der richtige Drucker für Ihre Anforderungen

TEXTVERARBEITUNG

- Gewußt wie:
  Tips zum Schreiben am PC
- Für Sie getestet:
  Die 10 populärsten TextProgramme
- Vergleichen & entscheiden:
  Das komplette Angebot im Überblick

GRAFIK

Das verständliche Computer-Magazin kommt:

- Klare Antworten.
- Bestechend einfache Problemlösungen.
- Kompetente Tests.

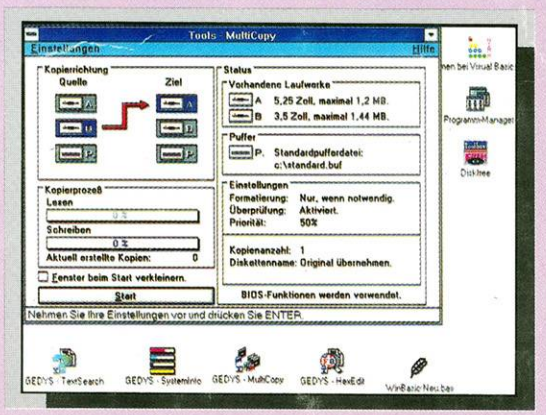
Für den Computeralltag!

Eine Publikation aus dem Hause DMV Daten- und Medienverlag

Ab 21. August im Zeitschriftenhandel!

DMV Windows Tools

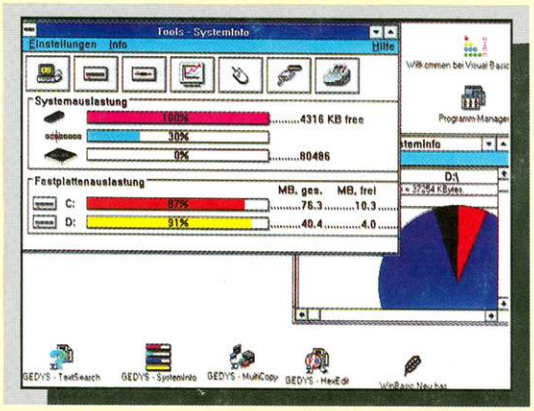
Wir machen Windows perfekt!



MULTYCOPY dupliziert Disketten superschnell und ohne daß immer wieder Ziel- und Quelldiskette gewechselt werden müssen

- nutzt erweiterten Speicher, RAM-Disk oder Festplatte als Puffer
- macht Mehrfachkopien von derselben Quelldiskette, auch zu späterem Zeitpunkt
- formatiert automatisch
- unterstützt Netzwerk
- **neu:** kopiert auch verschiedene Diskettenformate

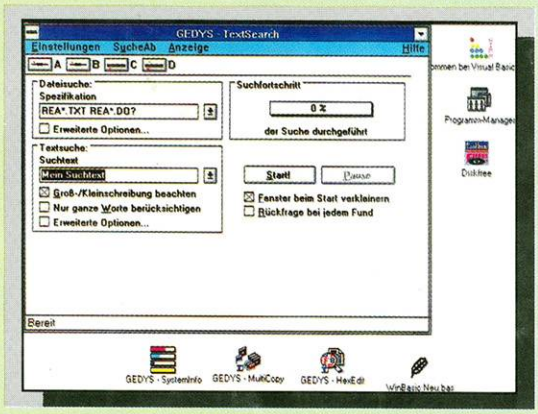
99,– DM*



SYSTEMINFO gibt Auskunft über Hardware und über die Systemauslastung zur optimalen Nutzung des Windows-Systems

- überwacht die Auslastung des Prozessors
- zeigt, wie der Hauptspeicher von Programmen verwendet wird
- zeigt die Speicher
- gibt grafisch Auskunft über Platzbelegungen
- listet die aktuelle Hardware-Konfiguration
- analysiert die Anschlüsse

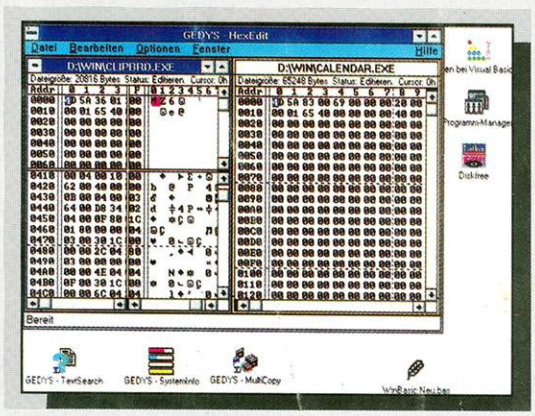
99,– DM*



TEXTSEARCH ist eines der schnellsten und leistungsfähigsten Such-Programme

- durchsucht Speichermedien nach gewünschtem Begriff
- läßt Kombination von Begriffen zu
- sucht global oder nach Pfadangaben
- akzeptiert Dateispezifikationen
- verarbeitet Wildcards
- listet Fundorte auf
- zeigt wahlweise das Textumfeld des Fundortes an
- sucht über ganzes Netz

99,– DM*



HEXEDIT zeigt Daten und Programme hexadezimal und zeichenorientiert

- Änderungen im Programmcode möglich
- Abspeichern möglich
- Suchen und Ersetzen von Text- und Bytefolgen
- Undo-Funktion
- ANSI- und IBM-Zeichensatz
- Bildschirmteilung horizontal und vertikal
- Dateigröße bis zu 64 MByte
- editiert mehrere Dateien gleichzeitig

99,– DM*

DMV-WINDOWS-TOOLS sind Produkte der Gedys Software Consulting GmbH

Systemanforderungen: IBM AT, IBM PS/2 ab Modell 30/286 und Kompatible, 1 MByte RAM, Festplatte, HGC, CGA, EGA, VGA, 8514/A, ab DOS 3.1 oder kompatiblen, MS-Windows 3.0.

Bitte benutzen Sie die Bestellkarte.

* Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland DM 4,– bzw. für das Ausland DM 6,– Porto und Verpackung.

DMV-Verlag • Postfach 250 • 3440 Eschwege

DMV Daten- und Medienverlag